Titre: Title:	Traitement et analyse d'images stéréoscopiques avec les approches du calcul générique sur un processeur graphique
Auteur: Author:	Ling Yun Ma
Date:	2012
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ma, L. Y. (2012). Traitement et analyse d'images stéréoscopiques avec les approches du calcul générique sur un processeur graphique [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/806/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/806/
Directeurs de recherche: Advisors:	Christopher J. Pal
Programme: Program:	Génie informatique

UNIVERSITÉ DE MONTRÉAL

TRAITEMENT ET ANALYSE D'IMAGES STÉRÉOSCOPIQUES AVEC LES APPROCHES DU CALCUL GÉNÉRIQUE SUR UN PROCESSEUR GRAPHIQUE

LING YUN MA
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
MARS 2012

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

TRAITEMENT ET ANALYSE D'IMAGES STÉRÉOSCOPIQUES AVEC LES APPROCHES DU CALCUL GÉNÉRIQUE SUR UN PROCESSEUR GRAPHIQUE

présenté par : MA, Ling Yun
en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées
a été dûment accepté par le jury d'examen constitué de :

M. LANGLOIS, J.M. Pierre, Ph.D., président
M. PAL, Christopher J., Ph.D., membre et directeur de recherche
M. BILODEAU, Guillaume-Alexandre, Ph.D., membre

À mes parents et mes sœurs,
merci pour vos encouragements...

REMERCIEMENTS

Je tiens d'abord à exprimer ma gratitude pour l'aide et la patience dont a fait preuve M. Christoper J. Pal, mon directeur de recherche. Je voudrais également remercier M. John Mullins, mon ancien directeur de recherche. C'est avec lui que j'ai entrepris mon travail de recherche au Canada.

Au cours de mes études à l'École Polytechnique de Montréal j'ai reçu aide et conseil de la part de mes professeurs, que je remercie vivement. Je voudrais souligner ma gratitude envers Michel Desmarais, Robert Roy et Jean-Pierre Crine pour leurs enseignements et conseils. Je remercie aussi Louise Longtin pour ses services pleins d'enthousiasme. Je remercie mes collègues de laboratoire, qui m'ont aidé dans la recherche, David Rim, Fannie Puech, Samira Ebrahimi Kahou, Md. Kamrul Hasan, Lucas Berthou. Merci également à Lam C. Tran, doctorant à l'Université de California, San Diego.

Je remercie Hélène Desrosiers et Alain-Marie Carron pour leurs appuis constants et leur confiance en moi. Ils ont consacré beaucoup de temps à la lecture et à la correction orthographique de ma thèse. Toutefois, les erreurs incluant celles de langue sont toutes sous ma pleine responsabilité. Je remercie également Chunling Ma, Sulun Yu, Shelly Chen, Sabrina Jiao, Jianfu Liu, Sophie Bisson et d'autres encore.

Enfin, et ce n'est pas le moins important, je remercie mes parents et mes sœurs pour leurs sacrifices et leur soutien indéfectible.

RÉSUMÉ

Étant donné une paire d'images stéréoscopiques, il est possible de calculer une carte de disparité dense qui encode les correspondances par pixel entre deux vues d'une même scène. Étant donné les paramètres de calibration d'une paire d'appareils photo, il est possible de transformer une carte de disparités en une carte de profondeur. Il existe de nombreuses applications allant de la robotique et de l'interaction humain-machine à la photographie 3D qui peuvent bénéficier de l'utilisation de cartes de disparité précises. Nous nous intéressons à la production de cartes de disparité de haute qualité à partir d'images stéréo pour des applications à temps réel ou dans un ordre de grandeur de temps réel par rapport au nombre d'images par seconde pour des vidéos typiques si un traitement hors ligne est acceptable. Nous avons donc étudié de possibilités d'accélérer de divers calculs clés nécessaires pour produire des cartes de disparité à partir d'images stéréoscopiques.

Tout d'abord, nous explorons le potentiel de détecter les disparités incompatibles avec un calcul rapide sur les images en basse définition et d'une vérification de consistance basée sur la comparaison entre une paire de cartes de disparité de gauche à droite et de droite à gauche. L'idée est que les disparités incompatibles sont susceptibles de contenir des erreurs. Puis nous évaluons le potentiel d'appliquer de calculs sélectifs en employant les images stéréoscopiques de plus haute définition afin de réduire les erreurs tout en évitant de calculs coûteux sur les images stéréoscopiques en plus haute définition tout entières. Nous avons aussi introduit une méthode d'interpolation simple et rapide qui est capable d'améliorer la qualité d'une carte de disparité si la densité de pixels consistants est élevée. Des travaux récents ont montré que la qualité d'une carte de disparité peut être améliorée en combinant différentes mesures de distance. Nous explorons une fonction de combinaison simple pour la somme des différences au carré et les distances de Hamming entre les blocs d'image représentés par la transformation Census. Nous montrons que cette technique de combinaison peut produire d'améliorations significatives quant à la qualité de carte de disparité. Nous explorons aussi des approches fondées sur la combinaison des deux mesures et la combinaison d'utilisation d'imageries en haute et basse résolutions de manière sélective.

Nous montrons aussi comment l'essence de méthodes populaires et d'état de l'art d'inférence semi-globale peut être formulée en utilisant des modèles de Markov cachés. Cela nous permet de généraliser les approches semi-globales à des modèles plus sophistiqués tout en ouvrant la porte aux paramètres des modèles d'apprentissage en utilisant des techniques du maximum de vraisemblance. Pour accélérer les calculs, normalement nous avons employé le calcul générique sur un processeur graphique (GPGPU). En particulier, nous avons implémenté en OpenCL

une variation de la mise en correspondance par bloc basée sur la somme des différences au carré et présenté une version corrigée de l'implémentation de l'algorithme Viterbi en OpenCL qui était fournie dans un kit de développement logiciel de GPU.

ABSTRACT

Given a pair of stereo images it is possible to compute a dense disparity map which encodes the per pixel correspondences between views. Given calibrated cameras it is possible to transform a disparity map into a depth map. There are many applications ranging from robotics and human computer interaction to 3D photography that benefit from the use of precise disparity maps. We are interested in producing high quality disparity maps from stereo imagery as quickly as possible for real-time applications or within an order of magnitude of real-time for typical video rates for applications where off-line processing is acceptable. We therefore explore the problem of accelerating various key computations needed to produce disparity maps from stereo imagery.

First, we explore the potential of detecting inconsistent disparities with fast but low resolution comparisons and a consistency check based on comparing left to right and right to left disparity maps. The idea is that inconsistent disparities are likely to contain errors. We then evaluate the potential of selectively applying computation using higher resolution imagery in order to reduce errors while avoiding expensive computations over the entire high resolution image. We also introduce a simple and fast interpolation method that is capable of improving the quality of a disparity map if the density of consistent pixels is high. Recent work has shown that disparity map quality can be also be improved by combining different distance metrics. We explore a simple combination function for sum of squared difference and Hamming distances between image blocks represented using the Census transform. We show that this combination technique can produce significant improvements in disparity map quality. We also explore approaches based on both combining metrics and selectively combining high and low resolution imagery.

We also show how the essence of popular, state of the art semi-global inference methods can be formulated using hidden Markov models. This allows us to generalize semi-global approaches to more sophisticated models while also opening the door to learning model parameters using maximum likelihood techniques. To accelerate computations generally we use general purpose graphical processing unit (GPGPU) computing. In particular, we have implemented a variation of sum of squared difference block matching in OpenCL and present a corrected version of an OpenCL Viterbi algorithm implementation that was provided in a GPU software development kit.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SIGLES ET ABRÉVIATIONS

2D	deux dimensions
3D	trois dimensions ou tridimensionnel, soit largeur, hauteur et profondeur
API	interface de programmation (*application programming interface*)
BilSub	soustraction de fond bilatéral (*bilateral background subtraction*)
BM	mise en correspondance par bloc (*block matching*)
CCN	corrélation croisée normalisée (*normalized cross correlation*)
CUDA	CUDA™, *compute unified device architecture*
DoG	différence de Gaussiennes (*difference of Gaussian*)
fps	images par seconde (*frames per second*)
GPGPU	calcul générique sur un processeur graphique (*general purpose GPU*)
GPU	processeur graphique (*graphic processor unit*)
HD	haute définition
HMI	information mutuelle hiérarchique (*Hierarchical Mutual Information*)
LED	diode électroluminescente (*Light-Emitting Diode*)
LoG	Laplacien de Gaussienne (*Laplacian of Gaussian*)
MMC	modèle Markov caché (*hiddem Markov model, HMM*)
MRF	modèle de champ aléatoire de Markov (*Markov random field*)
OpenCL	OpenCL™, *open computing language*
OpenCV	*open source computer vision library*
PE	élément de traitement (*processing element*)
RVB	rouge, vert et bleu
SDC	somme des différences au carré (*sum of squared difference, SSD*)
SDA	somme des différences absolues (*sum of absolute difference, SAD*)
SDK	kit de développement logiciel (*Software Development Kit*)
SGBM	*semi-global block matching*
SGM	*semi-global matching*
SIMT	seule instruction, multiples tâches (*single-instruction, multiple-thread*)
SM	streaming multiprocesseur
SSE2	*streaming SIMD extensions 2*
XOR	OU exclusif (*eXclusive OR*)

CHAPITRE 1

INTRODUCTION

Notre cerveau reçoit normalement des images similaires d'une scène des deux points proches (deux yeux) sur le même niveau horizontal. La différence relative de position des deux objets est appelée la disparité. Le cerveau est capable de mesurer la disparité et l'employer pour créer la sensation de profondeur. (voir Marr, 1982, p. 100) Si la disparité est mesurée en distance ou en nombre de pixels, la collection des disparités devient une carte de disparité. Le terme « distance » désigne la distance physique objective entre l'observateur et l'objet. Le terme « profondeur » est la distance subjective perçue par l'observateur vers l'objet. (voir Marr, 1982, p. 111) De la même façon, une collection de profondeurs est une carte de profondeur.

1.1 Carte de disparité

Une carte de disparité est une collection de distances de correspondance entre les pixels de deux images étudiées. Voici un exemple de carte de disparité avec à gauche deux images comme entrées dans la Figure 1.1. L'image de gauche et l'image de centre sont prises par l'appareil photo de gauche et l'appareil photo de droite respectivement, l'image de droite est la carte de disparité, qui est la collection de distance entre chaque pixel de l'image de gauche et le pixel correspondant de l'image de droite. Visuellement, l'objet plus proche aura une disparité plus élevée.

image de gauche d'*Art* | image de droite d'*Art* | carte de disparité de référence

Figure 1.1 Un exemple de deux images stéréoscopiques et une carte de disparité (voir Blasiak *et al.*, 2005)

La carte de disparité est liée étroitement avec la carte de profondeur. On peut créer une carte de profondeur à partir d'une carte de disparité si l'on connaît la longueur focale d'appareil photo en employant la méthode que nous allons introduire un peu plus loin dans la sous-section 1.2.

La carte de disparité est importante dans diverses applications reliées à la vue. Par exemple, une application d'augmentation de réalisme ou une application robotique pour calculer la distance entre le robot et l'objet. Diverses méthodes existent pour le calcul de disparité. Toutefois, la qualité de la carte de disparité et la performance du processus de création de cette carte constituent deux problèmes principaux. Nous avons étudié certains problèmes liés à la carte de disparité, évalué plusieurs moyens d'améliorer la qualité et la performance à l'aide de GPGPU.

Nous pouvons possiblement équiper une paire d'appareils photo avec un processeur pour produire des cartes de disparité en temps réel pour des applications qui demandent la haute vitesse de traitement. Nous pouvons aussi relier une machine de traitement à un appareil photo pour traiter rapidement les images capturées. Le traitement rapide est essentiel dans certaines applications comme la robotique, le jeu vidéo, etc. Maintenant les caméscopes numériques ne coûtent pas très cher. Si nous avons deux caméscopes numériques pour capturer des scènes, nous pouvons penser à produire une vidéo de disparité en temps réel.

Souvent nous n'avons pas besoin d'une carte de disparité de haute définition. Par exemple, une petite carte de profondeur produite par Kinect de Microsoft est assez précise pour rendre un jeu vidéo interactif. Pour alléger la transmission, nous pouvons traiter des photos prises en haute définition et envoyer la petite carte obtenue. Au lieu de transmettre des données lourdes de vidéo en haute définition, nous pouvons transmettre le résultat de traitement. En général, les méthodes de coûts globaux présentent toujours de meilleurs résultats. Toutefois, le temps de traitement est normalement élevé. Nous avons employé des méthodes simples pour mesurer le potentiel de l'emploi des images en haute définition.

Nous avons implémenté une méthode simple de mise en correspondance par bloc en OpenCL, puis étudié en détail le taux d'accélération, la qualité de carte et les limites de nos approches. Nous avons montré qu'il est possible de profiter des informations dans les images en plus haute définition pour améliorer la qualité de petites cartes de disparité. Une carte de disparité d'images en haute définition nécessite plus de ressources et est plus lente à calculer. L'utilisation de GPGPU peut accélérer le traitement, rendre cette méthode opérationnelle.

Il y a d'autres méthodes qui peuvent s'avérer très efficaces pour créer une grande carte de disparité. Par exemple, Geiger *et al.* ont présenté un bon résultat avec une approche de point de support et probabiliste. Les points de support sont obtenus avec l'algorithme de triangulation de Delaunay. (voir Geiger *et al.*, 2010).

Pour obtenir une carte continuelle et en améliorer la qualité, nous avons employé une interpolation. Nous avons aussi étudié BilSub (soustraction de fond bilatéral), MMC (modèle Markov caché), Census et une combinaison de Census et de mise en correspondance par bloc. Entre temps, nous avons aussi étudié les différences entre deux implémentations de l'algorithme Viterbi. Nous avons choisi la Méthode BilSub et la Méthode Census parce que selon Hirschmüller et Scharstein, la performance de BilSub est toujours très bonne pour les différences radiométriques basses, et la Méthode Census affiche la meilleure et la plus robuste performance globale (voir Hirschmüller et Scharstein, 2009).

1.2 Définitions et concepts de base

Brown *et al.* ont publié un article dans lequel la disparité est définie comme le déplacement de la position d'un point projeté dans une image par rapport à la position de ce même point dans l'autre image. L'ensemble de toutes les disparités entre deux images constitue une carte de disparité. (voir Brown *et al.*, 2003, p. 994) Nous allons présenter le concept et quelques méthodes.

1.2.1 Géométrie de stéréoscopie

Pour connaître la géométrie de stéréoscopie, nous devons d'abord choisir entre un système de stéréoscopie convergente et non convergente. Pissaloux *et al.* a montré une illustration dans la Figure 1.2 (voir Pissaloux *et al.*, 2008, p. 2)

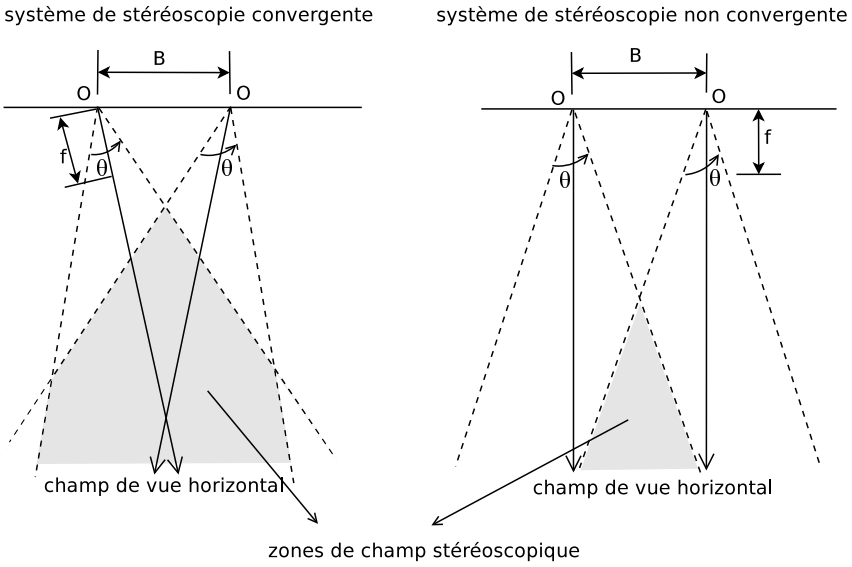


Figure 1.2 Géométrie de stéréoscopie convergente et non convergente (voir Pissaloux *et al.*, 2008, p. 2)

Le système de stéréoscopie convergente a un champ de vue horizontale moins large et plus proche que celui de stéréoscopie non convergente. Nous avons employé le système de stéréoscopie non convergente pour avoir un champ de vue plus large.

La relation entre deux appareils photo peut être calculée avec une transformation algébrique. Pour le système de stéréoscopie non convergente, les axes de coordination des deux appareils photo sont alignés et la transformation spatiale consiste en une translation d'une distance b dans l'axe x d'appareils photo. Une géométrie de stéréoscopie non convergente est montrée dans la Figure 1.3 où O_G et O_D sont les deux centres optiques. La distance entre O_G et O_D est T sur la ligne de base, f est la longueur focale des appareils photo, P_G et P_D sont deux images de P. La profondeur Z peut se calculer comme suit : $Z = f\frac{T}{d}$ où d est la disparité $d = x_G - x_D$ après la conversion en unité métrique. (voir Brown *et al.*, 2003, p. 994) La direction de vue est vers le haut au lieu de vers le bas comme dans la Figure 1.2.

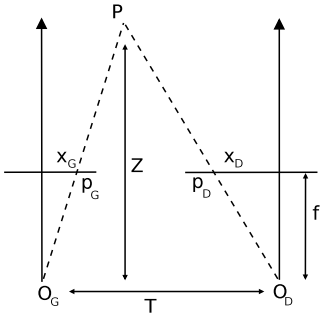


Figure 1.3 Géométrie de stéréoscopie non convergente (voir Brown *et al.*, 2003, p. 994)

La Figure 1.4 montre une autre illustration du calcul pour faciliter la compréhension. Cette fois, la projection d'image se trouve de l'autre côté de l'objet observé comme dans le modèle de sténopé. Le triangle image est formé en déplaçant virtuellement deux petits triangles T_G et T_D à deux bas côtés de la figure. Le triangle objet est en proportion avec le triangle image, donc nous pouvons avoir $\frac{Z}{T} = \frac{f}{x_G - x_D}$, alors $Z = f\frac{T}{x_G - x_D} = f\frac{T}{d}$ où d est la disparité $d = x_G - x_D$. C'est exactement le résultat ci-haut. (voir Thrun, 2011) Cette relation confirme que l'objet plus proche aura une disparité plus élevée.

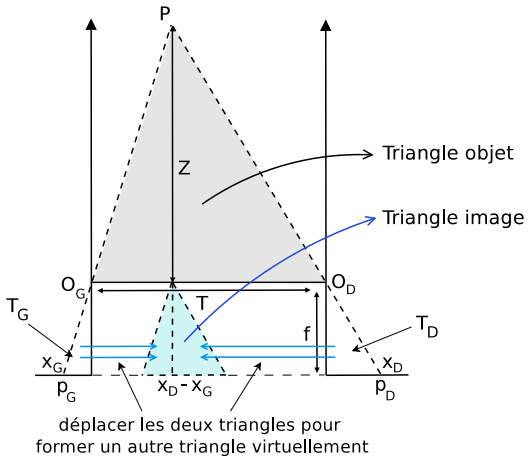


Figure 1.4 Géométrie de stéréoscopie non convergente (voir Thrun, 2011)

Il arrive souvent que nous n'ayons pas de système non convergent. Nous pouvons toutefois rectifier les images selon les contraintes épipolaires.

1.2.2 Mise en correspondance

La correspondance des pixels entre deux images est difficile à calculer. La raison principale de cette difficulté est due au fait que les deux images présentent diverses différences à cause des variations de lumière, d'intensité, d'appareils photo, d'occlusion et de texture. Les différences de lumière, d'appareils photo et d'intensité sont appelées des variations photométriques ou radiométriques. Une texture répétitive ou un manque de texture peut aussi créer des problèmes (voir Cochran et Medioni, 1992, p. 981). Le problème d'occlusion a lieu quand une partie d'une scène est visible pour un appareil photo, mais pas pour l'autre, normalement caché par une partie d'objet (voir Brown *et al.*, 2003, p. 1002).

La correspondance peut se faire par la correspondance des caractéristiques spécifiques, par exemple, des coins des deux images, ou par la correspondance des petites régions par corrélation sans identifier de caractéristiques. La correspondance des caractéristiques spécifiques peut être plus efficace que la correspondance par petites régions, mais limitée dans l'application parce qu'il est difficile de trouver une caractéristique fiable. La correspondance par petites régions peut être plus facile à employer, mais la recherche est coûteuse. Deux mesures de similarité sont employées souvent entre les régions : la corrélation croisée et la somme des différences au carré (SDC). (voir Nevatia, 1982, p. 160)

La correspondance des caractéristiques spécifiques peut produire une carte de disparité avec des points fiables, le résultat est une carte de disparité clairsemée. On a normalement besoin d'une interpolation pour compléter la carte si une carte dense est demandée. Une carte

dense calcule la disparité de chaque pixel au lieu de pixels principaux dispersés. Dépendant de la méthode d'interpolation, cette méthode peut avoir plus de chance d'ignorer de détails. La correspondance par petites régions peut efficacement produire une carte dense, c'est pourquoi cette méthode est devenue la méthode principalement étudiée dans ce mémoire. Malheureusement, elle a de difficulté à traiter des zones homogènes. Nous allons essayer de méthodes pour contrer ce problème.

Comme méthode supplémentaire, la méthode de vues multiples et la méthode de correspondance d'une vue d'ensemble à une vue plus précise sont deux simplifications de recherche de correspondance.

1.2.3 Méthode de vues multiples

La méthode de vues multiples permet de limiter la gamme de recherche de correspondance par la division de 2 vues initiales en plusieurs vues. Une limite de gamme de correspondance peut augmenter la précision sans augmenter le temps de recherche. Les disparités entre les deux vues extrêmes sont le résultat après le chaînage des vues intermédiaires. (voir Nevatia, 1982, pp. 161–162) La Figure 1.5 montre l'effet de cette méthode. Pour une même largeur de champ de stéréoscopie, un ajout du troisième appareil photo O_c entre O_g et O_d peut couper le champ en deux : le champ 1 et le champ 2, où chacun a une étendue réduite de disparités. En conséquence, les vues peuvent être prises plus proches des appareils photo, ce fait peut aussi encore aider à augmenter la précision. La direction de vue est vers le haut comme dans la Figure 1.3.

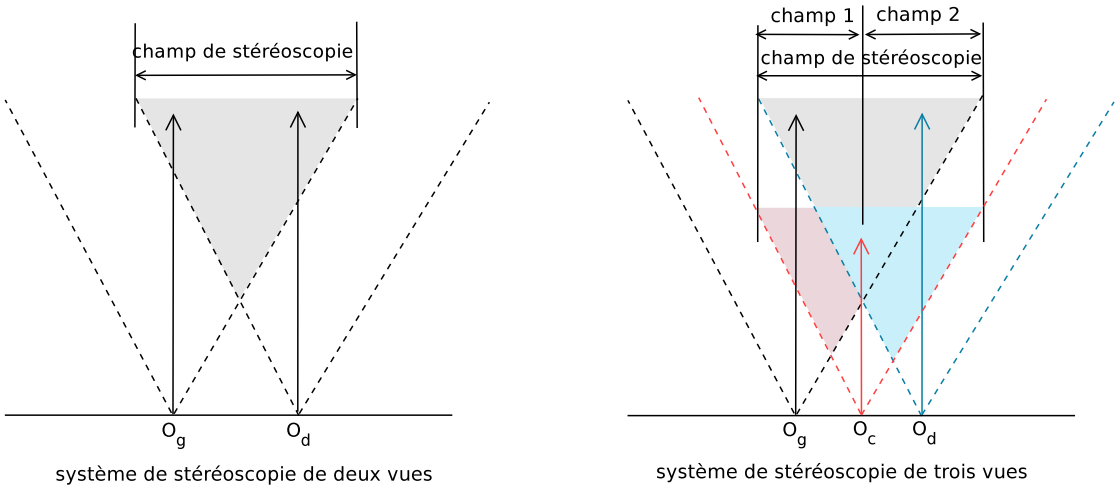


Figure 1.5 Géométrie de stéréoscopie non convergente et de vues multiples

1.2.4 Méthode de correspondance d'une vue d'ensemble à une vue plus précise

Quant à la méthode de correspondance d'une vue d'ensemble à une vue plus précise, plusieurs auteurs incluant Moravec et Cochran et Medioni ont présenté des méthodes basées sur la correspondance d'une vue d'ensemble à une vue plus précise. La correspondance d'une vue d'ensemble à une vue plus précise entraîne une perte d'information au niveau d'images réduites. Cette perte affecte la précision de correspondance (voir Nevatia, 1982, p. 162). Moravec a employé un « opérateur d'intérêt (*Interest Operator*) » pour mesurer la correspondance d'une vue d'ensemble à une vue plus précise (voir Moravec, 1980). Cochran et Medioni ont pour leur part présenté une méthode qui emploie des images en pyramide. Dans cette méthode, une estimation des disparités des niveaux généraux guide la recherche de correspondance vers des niveaux plus précis. Le travail est basé sur une mise en correspondance par bloc (voir Cochran et Medioni, 1992).

1.2.5 Filtrage

Deux images prises de deux appareils photo peuvent présenter des variations dues à la différence d'appareils photo, à une mauvaise calibration d'appareils photo ou à la différence de lumière. Par exemple, Hirschmuller et Gehrig ont employé un filtre Sobel sur la direction x s'appelant XSobel, le noyau de convolution est : (voir Hirschmuller et Gehrig, 2009, p. 2)

$$S_x = \frac{1}{4}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \tag{1.1}$$

Le filtre XSobel peut supprimer les fréquences basses autour de l'axe f_y pour éliminer les variations à cause de problèmes de calibration d'appareils photo. Les auteurs ont également proposé un filtre HBilSub qui est une modification de BilSub (soustraction de fond basé sur le filtre bilatéral) que nous allons présenter plus tard. Le filtre HBilSub est : (voir Hirschmuller et Gehrig, 2009, p. 3)

$$\begin{aligned} I_f(x,y) &= I(x,y) - \frac{\sum_{x' \in N_x} I(x,y) e^s e^r}{\sum_{x' \in N_x} e^s e^r} \\ s = -\frac{(x'-x)^2}{2\sigma_s^2}, r &= -\frac{(I(x',y)-I(x,y))^2}{2\sigma_r^2} \end{aligned} \tag{1.2}$$

Il existe aussi plusieurs méthodes basées sur l'utilisation d'un filtre qui permet de lisser les images à traiter ou même, dans certains cas, la carte de disparité. Ces méthodes sont efficaces, mais pas dans tous les cas. Par exemple, un filtre destiné à éliminer la bande

de haute fréquence peut avoir des problèmes dans les bordures d'objets des images. Les paramètres associés au filtre dépendent aussi des types d'images à traiter.

Nous avons étudié le filtre bilatéral et une méthode d'amélioration de qualité de correspondance basée sur le filtre bilatéral s'appelant BilSub.

1.2.6 Interpolation

Il y a souvent des zones où on ne dispose pas de valeurs fiables de disparité. C'est souvent le cas dans les zones lisses sans caractéristique spécifique ou dans les bordures d'occlusion. Toutefois, une carte de disparité doit être complète et ne pas comporter de trou. Pour cela, nous pouvons employer l'interpolation pour propager des informations et conserver la discontinuité de disparité. (voir Fua, 1991, p. 1296)

Nous pouvons aussi calculer une carte de disparité obtenue avec des points clés des images stéréoscopiques, et employer l'interpolation pour combler le reste. Normalement nous avons besoin de segmenter les images stéréoscopiques afin de mieux remplir une carte de disparité parsemée. On peut remplir la zone avec une fonction linéaire à l'intérieur d'un segment.

Nous avons introduit une interpolation simple pourtant efficace s'appelant MoyenPlusPropagationDuFond qui propage d'information à partir des disparités déjà obtenues. Évidemment, les disparités initiales doivent être assez fiables.

1.3 Éléments de la problématique

Si on calcule une carte de disparité avec de petites images, toutes les méthodes existantes ont des limites. Les méthodes dépendant des caractéristiques spécifiques des images ne peuvent pas s'adapter à tous les types d'images. Si on emploie la mise en correspondance par bloc pour augmenter la stabilité de correspondance, on risque de déplacer les bordures horizontalement et en même temps d'élargir les zones à gauche des bordures. Le manque d'information est la source principale de ces problèmes.

Si on utilise des images en haute définition, les méthodes existantes sont pour la plupart très coûteuses en temps. Trouver le moyen d'accélérer le processus devient une question importante. D'une part, les images en haute définition fournissent plus de détails. D'autre part, la variation locale peut éventuellement dévier la disparité. Il faut élaborer des méthodes qui permettent d'employer efficacement les images en haute définition.

1.4 Objectifs de recherche

Aujourd'hui, les processeurs sont plus puissants, les appareils photo ont une plus haute définition pour un prix moins élevé. Nous voulons employer des méthodes simples, en com-

binaison avec quelques techniques pour créer une meilleure carte de disparité.

Les objectifs de la recherche sont d'évaluer et de proposer des méthodes pour minimiser le temps de traitement et pour maximiser la qualité de la carte de disparité, incluant :

- Étudier la possibilité de rendre la carte de disparité plus précise par l'emploi d'images en plus haute définition et l'impact de l'emploi des images HD ;
- Rendre la création de la carte de disparité proche du temps réel sur un même ordinateur par l'emploi de GPGPU ;
- Évaluer le potentiel d'employer MMC pour l'amélioration de la qualité de carte de disparité ;
- Évaluer la possibilité d'employer de méthodes de calcul sélectif pour améliorer la performance.

Nous allons étudier plusieurs solutions potentielles et montrer des méthodes possibles. Nous voulons examiner chaque étape du processus de la création de carte de disparité pour connaître au maximum le potentiel d'amélioration à chaque étape afin de trouver une méthode efficace pour traiter les images en haute définition.

Afin de résoudre les contraintes de performance liées aux images en haute définition, nous allons employer GPGPU dans le traitement. Pour ce qui est des méthodes d'amélioration de qualité, nous allons étudier Census, combinaison de Census et de mise en correspondance par bloc, nous allons également étudier les méthodes du modèle Markov caché et de la combinaison de Census et de MMC pour voir le potentiel.

Nous rencontrons souvent des scènes similaires, ou d'environnements contrôlés. Nous voulons connaître la possibilité d'employer un groupe de matrices de transition et d'émission pour créer des cartes de disparité de qualité dans chaque type de ces situations. Pour cela, nous allons implémenter une application MMC pour étudier le potentiel d'accélération de modèle probabiliste. La Figure 1.6 montre le modèle de Markov caché. Les points gris sont les observations, les points ronds vides sont les variables cachées que nous voulons connaître.

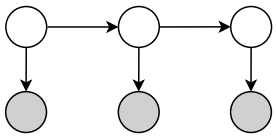


Figure 1.6 Modèle de Markov caché

Le modèle MMC est appliqué de ligne en ligne, ou de colonne en colonne, il ne prend en compte les caractéristiques du voisinage que dans une seule direction. Le résultat montre qu'il y a souvent de petites bandes isolées. Pour améliorer la situation, nous pouvons appliquer un

modèle composé de deux ou plusieurs directions. La Figure 1.7 montre un modèle de Markov caché (MMC) en grille carrée et un modèle MMC en deux étapes. Les points gris sont les observations, les points ronds vides sont les variables cachées que nous voulons connaître.

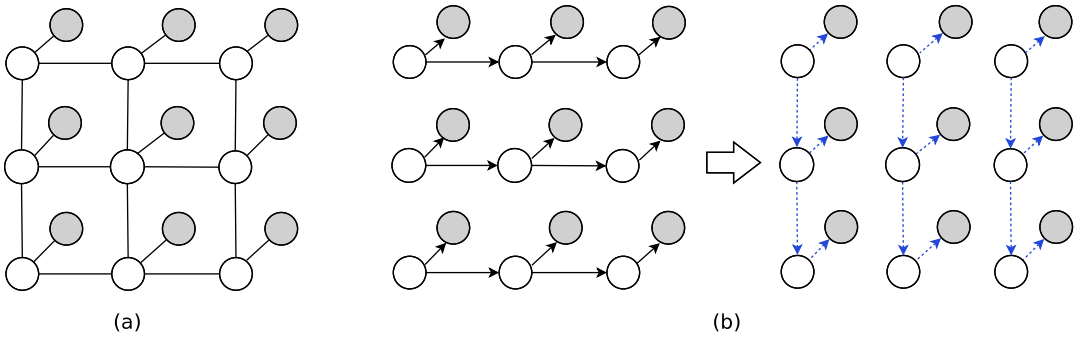


Figure 1.7 Une généralisation d'un modèle de Markov caché (MMC) en grille carrée et un MMC en deux étapes

Le modèle (a) de la Figure 1.7 est un modèle populaire et standard de fonction d'interférence d'énergie globale. Normalement on emploie la méthode de propagation de croyance (voir Felzenszwalb et Huttenlocher, 2004; Yang *et al.*, 2009) ou de coupes de graphes (voir Kolmogorov et Zabih, 2001) pour calculer. Dans le cas de propagation de croyance comme méthode probabiliste, le temps requis et le mémoire pour sauvegarder les messages sont normalement élevés. Il est compliqué de faire d'apprentissage ou d'employer ce modèle si on demande le temps réel. Nous pouvons essayer le modèle (b) de la Figure 1.7. Nous allons essayer d'employer le MMC en deux ou plusieurs étapes pour le rapprocher. L'algorithme Viterbi employé dans le MMC est un algorithme de programmation dynamique. Les modèles de la Figure 1.7 ressemblent beaucoup à la méthode de mise en correspondance semi-globale (SGM). Nous essayons d'inclure le plus possible d'information de voisinage dans l'interférence semi-globale. L'avantage de MMC est qu'il nous permet d'apprendre les paramètres à partir des données pour mieux approcher la carte de disparité idéale. Le modèle (b) est simple de faire l'apprentissage en temps réel.

Nous allons également essayer des combinaisons de plusieurs méthodes basées sur le MMC, incluant une combinaison de mise en correspondance par bloc et de MMC, et une combinaison de Census et de MMC. Le MMC peut aider à faire la correspondance en zones sans texture et pour les environnements similaires.

Comme une méthode supplémentaire, nous allons aussi étudier la possibilité de faire une application stéréoscopique de visage en temps réel par une méthode de projection de lumière colorée.

1.5 Plan du mémoire

Nous avons d'abord introduit quelques éléments de base dans les études de stéréoscopie. Par la suite dans le Chapitre 2 nous avons introduit le GPGPU qui est une ressource indispensable dans nos études. Nous avons également fourni des notes sur la façon d'augmenter la performance. Comme début du sujet principal de recherche, nous avons fait une revue de littérature à propos de quelques techniques qui nous intéressent dans le Chapitre 3. Dans le Chapitre 4, nous avons introduit quelques algorithmes implémentés sur OpenCV. Comme la partie principale, nous avons étudié la possibilité d'employer les images en haute définition pour créer une carte de disparité et la création d'une carte de disparité avec une approche simple de mise en correspondance par bloc. Nous avons également montré le potentiel de BilSub. Nous avons montré que les images en haute définition peuvent bel et bien contribuer à créer une carte de disparité plus précise. Afin de créer une carte de disparité sans trou, nous avons employé une méthode simple d'interpolation pour remplir des zones avec des trous.

Pour améliorer encore la carte de disparité, nous avons introduit plusieurs méthodes dans le Chapitre 5. Nous avons étudié la méthode Census pour chercher son potentiel. Pour continuer l'exploit de la méthode de remplissage, nous avons montré l'utilité de l'emploi des images en plus haute définition et l'interpolation. Ensuite nous avons proposé la combinaison de Census et la correspondance, montré de bons résultats avec un certain choix de fonction de combinaison.

Nous avons employé le MMC après la première étape de création de carte, étudié le potentiel de cette approche et l'effet de plusieurs passes de MMC. Nous avons aussi étudié la combinaison de mise en correspondance par bloc et de MMC, la combinaison de Census et de MMC. Ensuite nous avons montré les comparaisons de la performance en chiffre sur les différentes images. Comme une méthode possible pour la mise en correspondance par bloc, nous avons projeté de la lumière structurée sur l'objet étudié, montré le bon résultat avec la mise en correspondance par bloc en mode couleur RVB.

Au cours de nos expériences, nous avons étudié deux implémentations disponibles de Viterbi, et corrigé un code du SDK de NVIDIA®. Nous avons expliqué ces différences et fourni en détail la correction d'un code du SDK de NVIDIA® en complément.

CHAPITRE 2

GPGPU

GPU (*Graphic Processor Unit*) est spécialisé pour le calcul intensif et hautement parallèle parce qu'il possède plus de transistors dans le traitement de données. La Figure 2.1 montre clairement cette différence. (voir NVIDIA, 2011b, p. 3)

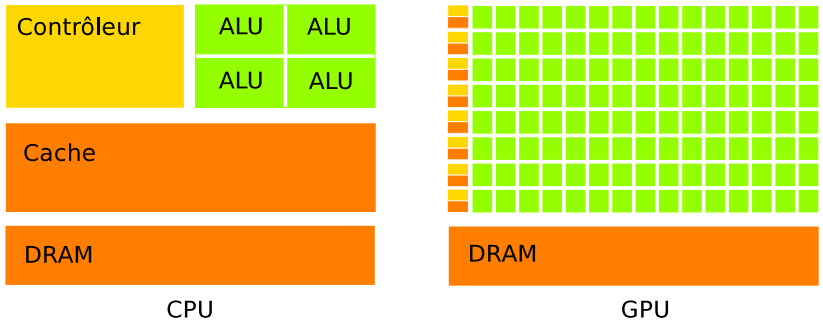


Figure 2.1 Illustrations des structures de CPU et de GPU (voir NVIDIA, 2011b, p. 3)

La Figure 2.2 montre un *die* de processeur Core i7 d'Intel® qui a quatre noyaux de CPU avec multithreading simultané, 8 Mo cache L3, contrôleurs DRAM sur puce. Fabriqué avec une technologie des procédés de 45 nm, chaque puce a 731 millions transistors et consomme jusqu'à 130 W de puissance de conception thermique. Les parties avec un rectangle en rouge soulignent la portion d'unité d'exécution de chaque noyau. Les images sont venues d'Intel inc. sauf les lignes en rouge. (voir Glaskowsky, 2009, p. 5)

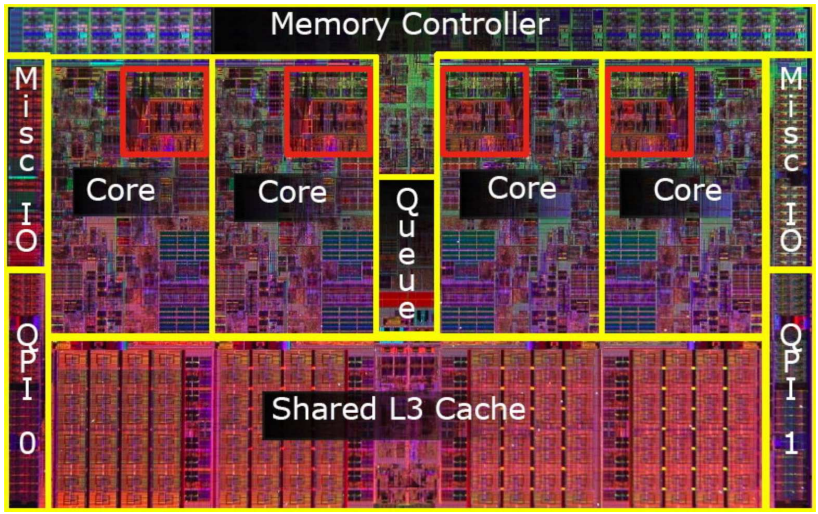


Figure 2.2 *Die* de Core i7 processeur d'Intel (voir Glaskowsky, 2009, p. 5)

Nous pouvons voir que seule une petite partie de la puce est employée pour faire les calculs. Tandis que GPU consacre la plupart de ses ressources aux calculs.

Le concept GPGPU (*General Purpose GPU*) consiste à employer le pouvoir de traitement parallèle de GPU dans un usage général. Pour des applications qui demandent des opérations sur les données massives qui peuvent être exécutées parallèlement, cela peut augmenter de beaucoup la performance par rapport au CPU conventionnel. On emploie GPU pour désigner aussi GPGPU.

La Figure 2.3 affiche la nouvelle architecture Fermi de GPU de NVIDIA®. Un streaming multiprocesseur (SM) exécute un ou plusieurs blocs de fils d'exécution. 16 streaming multiprocesseurs se positionnent autour d'un cache L2 commun. Chaque SM contient un contrôleur d'ordonnancement (orange), des unités d'exécution (vert), et des fichiers de registre et cache L1 (bleu clair). (voir NVIDIA, 2009, p. 7)

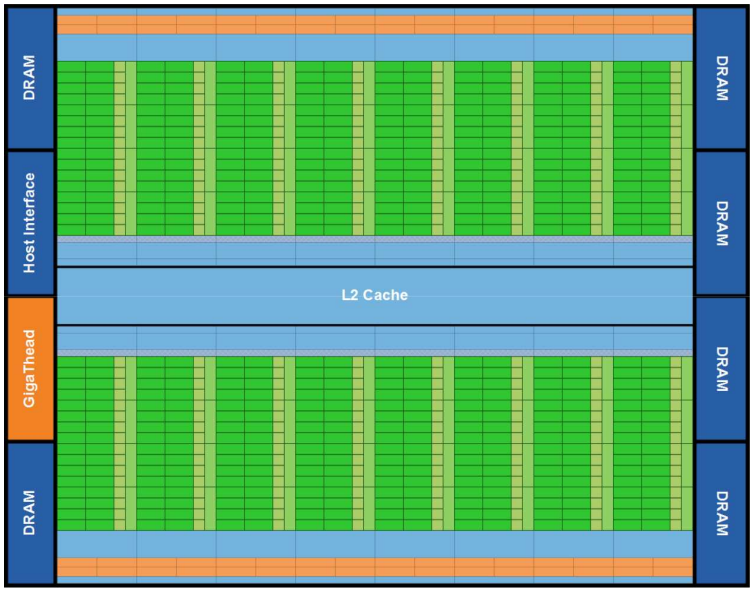


Figure 2.3 Architecture de Fermi (voir NVIDIA, 2009, p. 7)

De la figure d'architecture de GPGPU, on peut comprendre que pour accélérer il faut moins d'embranchements et il faut laisser les noyaux du processeur travailler autant que possible. Nos expériences ont prouvé que des commandes comme
__syncthreads() de CUDA et barrier(CLK_LOCAL_MEM_FENCE) de OpenCL et break peuvent toutes ralentir l'exécution. Donc, il faut soit éviter ces commandes, soit les limiter si elles sont incontournables.

Si on regarde en détail la Figure 2.4, chaque Fermi SM a 32 noyaux, 16 unités de chargement/sauvegarde, 4 unités de fonction spéciale, un fichier de registre de 32 K mots, 64 K RAM configurable et contrôleur de fils d'exécution. Chaque noyau possède des unités d'exécution à la fois en entier et en virgule flottante. (voir Glaskowsky, 2009, p. 20)

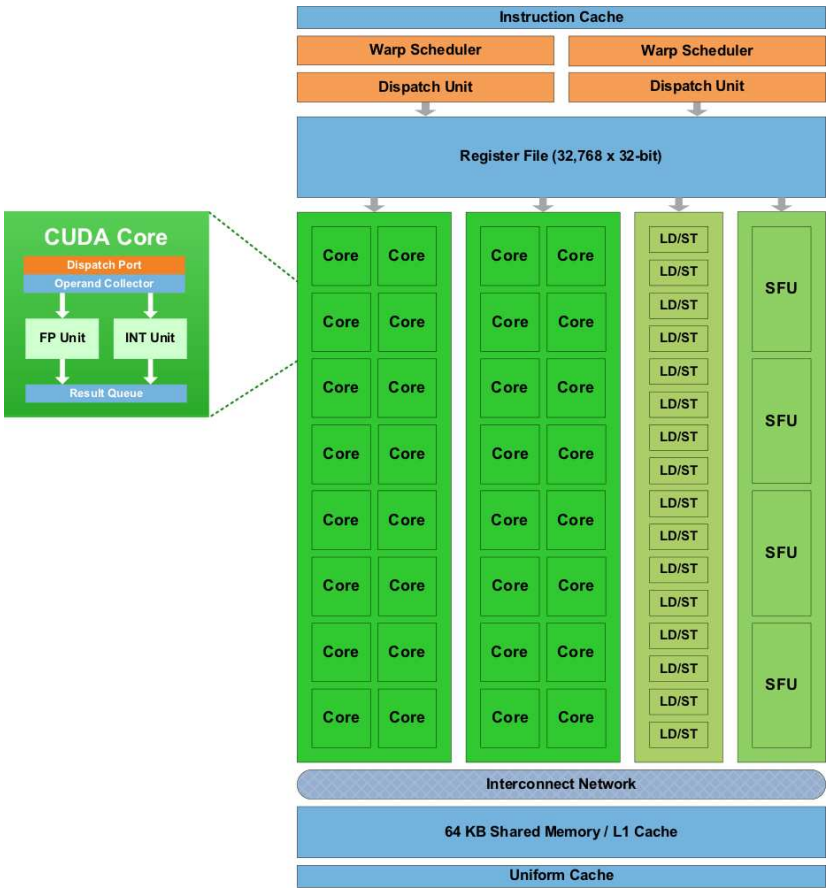


Figure 2.4 SM dans l'architecture Fermi (voir Glaskowsky, 2009, p. 20)

Nous allons présenter quelques points importants sur CUDA C, OpenCL et la façon d'améliorer la performance avec GPGPU. La raison pour laquelle nous présentons aussi CUDA C bien que nous travaillons principalement avec OpenCL est que nous employons des GPGPU de NVIDIA®, et que des guides, des principes de programmation et des notions sont souvent interchangeables sur GPGPU de NVIDIA®.

Dans ce document, un dispositif désigne une collection d'unités de calcul. (voir Munshi, 2009, p. 14) Nous présentons ici quelques notes d'utilisation de GPGPU.

2.1 CUDA C

La loi d'Amdahl spécifie que l'on peut évaluer le taux maximal d'accélération S espérée par la portion d'un programme séquentiel pouvant être rendue parallèle.

$S = \frac{1}{(1-P)+\frac{P}{N}}$ où P est la fraction de code pouvant être rendue parallèle. N est le nombre de processeurs que la portion parallèle peut employer. On peut voir que la meilleure pratique pour accélérer est de maximiser la partie de code pouvant être exécutée en parallèle. (voir NVIDIA, 2011a, p. 7)

CUDA est une architecture de calcul parallèle rendue publique par NVIDIA® en novembre 2006. CUDA supporte plusieurs langages ou interfaces de programmation d'application, incluant C, FORTRAN, OpenCL et DirectCompute. (voir NVIDIA, 2011b, p. 2) La plus petite unité d'exécution de parallélisme sur CUDA™ contient 32 fils (un *warp*). (voir NVIDIA, 2011a, pp. 4–5)

Le composant d'exécution d'hôte de l'environnement CUDA fournit les fonctions ci-dessous. Ces fonctions ne peuvent être employées que par les fonctions d'hôte.

- Gestion de ressource
- Gestion de contexte
- Gestion de mémoire
- Gestion de module de code
- Gestion d'exécution
- Gestion de référence de texture
- Interopérabilité avec OpenGL et Direct3D

Il contient 2 APIs :

- Un API de bas niveau : CUDA API de pilote
- Un API de haut niveau : CUDA API d'exécution

Les deux APIs peuvent se distinguer facilement. L'API de pilote est livrée par la bibliothèque dynamique nvcuda / libcuda et tous les points d'entrée sont préfixés avec cu. Alors que l'API d'exécution est livré par la bibliothèque dynamique cudart et tous les points d'entrée

sont préfixés avec cuda. L'API d'exécution pour la plupart des cas est préférable. (voir NVIDIA, 2011a, pp. 10–12)

Pour désigner un type de dispositif, on emploie la notion de capacité de calcul. La capacité de calcul d'un dispositif est définie par un numéro de révision majeure et un numéro de révision mineure. Un dispositif avec le même numéro de révision a la même architecture de noyau. Le numéro de révision majeure des dispositifs basés sur l'architecture Fermi est 2. Les dispositifs avant l'architecture Fermi ont tous la capacité de calcul 1.x. (voir NVIDIA, 2011b, p. 14)

2.1.1 Quelques méthodes d'améliorer la performance

CUDA supporte la mémoire épinglée (*pinned memory*), le transfert asynchrone et le chevauchement de transfert avec calcul pour améliorer la performance. La mécanique de mémoire épinglée permet de réserver une partie de la mémoire vive hôte à être accédées par le dispositif pendant une exécution en employant les appels cudaMallocHost() ou cudaHostAlloc(). (voir NVIDIA, 2011a, pp. 21–22)

Le support pour permettre aux fils de GPGPU d'accéder à la mémoire d'hôte peut encore accélérer dans certains cas, en particulier pour GPGPU intégré parce que le GPGPU et le CPU partagent la même mémoire. (voir NVIDIA, 2011a, pp. 21–22)

La clé pour améliorer la performance est d'éliminer le transfert lent, de bien employer la mémoire partagée, d'optimiser l'emploi d'instruction et de rendre parallèle le programme autant que possible ou de garder les noyaux aussi occupés que possible.

La mémoire partagée contient les paramètres ou les arguments qui sont transmis aux noyaux au lancement. Pour les noyaux qui chargent de longues listes d'arguments, il peut être utile de mettre quelques arguments dans la mémoire constante (et de les référencer là) plutôt que de consommer de la mémoire partagée. (voir NVIDIA, 2011a, p. 38)

La mémoire locale a une portée locale au fil d'exécution. Elle n'a pas d'accès rapide. L'accès à la mémoire locale est aussi coûteux que l'accès à la mémoire globale. (voir NVIDIA, 2011a, p. 38)

L'accès à la mémoire globale avec l'emploi de texture est plus rapide pour les dispositifs de capacité de calcul 1.x. En particulier pour les cas d'arrangement en mémoire qui n'est pas optimal. Toutefois, le cache L1 des dispositifs de capacité de calcul 2.x a une bande passante plus haute que celle du cache de texture. Donc, il est possible que cet avantage n'en soit pas en certains cas. (voir NVIDIA, 2011a, p. 39)

Une métrique pour déterminer le nombre d'unités de fils d'exécution (*warp*) actifs s'appelle le taux d'occupation. Cette métrique est en même temps le ratio entre le nombre d'unités de fils d'exécution par multiprocesseur et le nombre maximal d'unités de fils d'exécution pos-

sibles. Un facteur important de taux d'occupation est la disponibilité de registre. Toutefois, une fois qu'un taux d'occupation de 50% a été atteint, une augmentation supplémentaire d'occupation ne se traduira pas en une performance améliorée. (voir NVIDIA, 2011a, pp. 44–45)

2.1.2 Support différent d'arithmétiques entières

Sur les dispositifs de capacité de calcul de 1.x, la multiplication entière de 32-bit est implémentée avec plusieurs instructions parce qu'elle n'est pas nativement supportée. La multiplication entière de 24-bit est nativement supportée. Par contre, l'inverse est vrai sur les dispositifs de capacité de calcul de 2.x. La multiplication entière de 32-bit est nativement supportée. La multiplication entière de 24-bit est implémentée avec plusieurs instructions. (voir NVIDIA, 2011b, p. 100)

Comme partout, l'opération de décalage est préférable aux calculs de division et de modulo (voir NVIDIA, 2011a, p. 50). Il est intéressant de noter qu'un entier signé est préférable à un entier non signé comme compteur de boucle. Ceci est dû à la capacité du compilateur d'optimiser dans le cas d'entier signé. (voir NVIDIA, 2011a, p. 56)

2.1.3 Compilations

Un programme CUDA C est compilé avec `nvcc` vers une forme d'assembleur (code PTX) et/ou forme binaire (objet `cubin`). Tout code PTX chargé par une application à l'exécution est compilé dans un deuxième temps en code binaire par le pilote de dispositif. C'est ce qu'on appelle compilation juste à temps. Une compilation juste à temps augmente le temps de chargement d'application, mais permet aux applications de bénéficier des dernières améliorations du compilateur. (voir NVIDIA, 2011b, p. 16)

2.1.4 Accès directs

Pour accélérer, il est possible d'accéder à la mémoire en mode pair à pair, et d'employer un espace d'adressage virtuel unifié pour des dispositifs de capacité de calcul de 2.x et de série Tesla. (voir NVIDIA, 2011b, p. 36–37)

2.1.5 Textures en série

Les dispositifs de capacité de calcul de 2.x supportent les textures en série, ou un tableau de textures (voir NVIDIA, 2011b, p. 44). Cette fonction peut possiblement faciliter l'interpolation de plusieurs images en série et de différentes résolutions. Parce que les mémoires de texture et de surface sont mises en cache, et que le cache n'est pas cohérent par rapport

à l'écriture de la mémoire globale et de la mémoire de surface, une lecture de texture ou une lecture de la surface d'une adresse qui ont été écrites à travers une écriture globale ou une écriture de surface dans le même appel de noyau, retourneront des données non définies. En conséquence, une adresse de texture ou de surface peut être lue en sécurité seulement si l'adresse a été mise à jour par un appel de noyau précédent ou une copie de mémoire. La mémoire de surface est un tableau de CUDA créé avec une étiquette cudaArraySurfaceLoadStore, peut être lue ou écrite par le biais de la référence de surface en employant des fonctions fournies. La différence entre la mémoire de texture et celle de surface est que cette dernière emploie l'adressage d'octets au lieu de l'adressage d'éléments de la mémoire de texture. (voir NVIDIA, 2011b, p. 45–47)

2.1.6 Interopérabilités graphiques

Quelques ressources d'OpenGL et de Direct3D peuvent être mappées dans l'adresse de CUDA pour échanger des données. (voir NVIDIA, 2011b, p. 46)

2.1.7 SIMT

Multiprocesseur emploie une architecture SIMT (Seule Instruction, Multiples Tâches, *Single-Instruction, Multiple-Thread*). Une seule instruction contrôle de multiples fils d'exécution. S'il y a une divergence dans une branche, les autres fils doivent attendre la fin de l'exécution d'un fil en particulier. (voir NVIDIA, 2011b, pp. 85–86)

2.1.8 Accès de mémoire de dispositif

La mémoire globale et la mémoire locale résident dans la mémoire des dispositifs. La mémoire partagée résidant dans la puce, l'accès sera plus rapide que la mémoire globale et la mémoire locale. (voir NVIDIA, 2011b, pp. 93–96)

2.1.9 Contrôles de processus

Il y a des fonctions de barrière de mémoire et des fonctions de points de synchronisation pour s'assurer le bon fonctionnement du processus (voir NVIDIA, 2011b, pp. 113–115).

2.2 OpenCL

OpenCL™ (*Open Computing Language*) est un standard libre pour une programmation parallèle à usage général à travers CPU, GPU et d'autres processeurs (voir Munshi, 2009,

p. 11). On peut écrire des programmes portables qui s'exécutent sur différents dispositifs et architectures.

"Le langage de programmation OpenCL est basé sur la spécification du langage C ISO/IEC 9899 :1999 avec certaines extensions et restrictions." (voir Munshi, 2009, p. 126)

2.2.1 Compilation de noyau

Le processus de compilation d'un noyau est :

1. Créer un programme : l'entrée est un code source ou un code binaire précompilé.
2. Compiler le programme : spécifier le dispositif de destination et passer les spécifications.
3. Créer un noyau : un objet noyau sera retourné.

2.2.2 Comparaisons entre CUDA C et OpenCL

Nous avons comparé certains points entre CUDA C et OpenCL ci-dessous. Pour plus d'information, Rosendahl a présenté une comparaison entre CUDA et OpenCL (voir Rosendahl, 2010).

- CUDA est une technologie propriétaire de NVIDIA. Ses outils et SDK sont fournis gratuitement par NVIDIA®. OpenCL est un standard industriel ouvert pour programmer une collection hétérogène de CPU, GPU et d'autres dispositifs discrets de calcul dans une seule plateforme. OpenCL est ouvert, libre de redevances, initié par Apple inc., sa spécification est maintenue par le groupe Khronos™. OpenCL est une marque de commerce d'Apple inc., employée sous licence par Khronos. Ses outils et SDK sont fournis par des fournisseurs de matériaux.
- CUDA supporte l'intégration profonde de programmes d'hôte et de dispositifs. Dans OpenCL, le code noyau est livré avec le code binaire d'application.
- Le modèle CUDA est par nature une architecture NVIDIA® orientée. Le modèle OpenCL est plus générique, aussi emploie-t-il une terminologie plus générique. OpenCL ne supporte que la compilation séparée et une invocation de noyau par le biais d'appel API.

La Figure 2.5 présente l'espace mémoire dans un dispositif CUDA (voir NVIDIA, 2011a, p. 24).

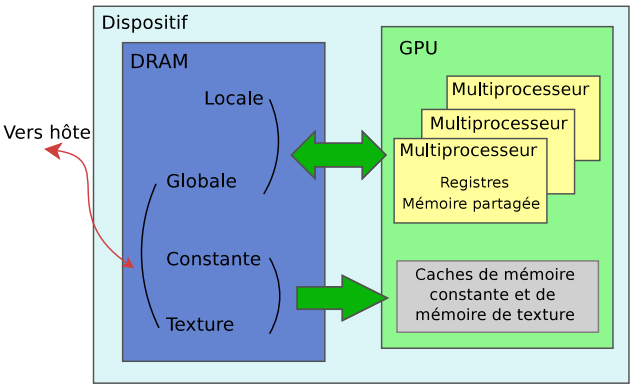


Figure 2.5 Espace mémoire dans un dispositif CUDA (voir NVIDIA, 2011a, p. 24)

La Figure 2.6 présente l'architecture du dispositif d'OpenCL. PE signifie un élément de traitement (*Processing Element*) (voir Munshi, 2009, p. 25).

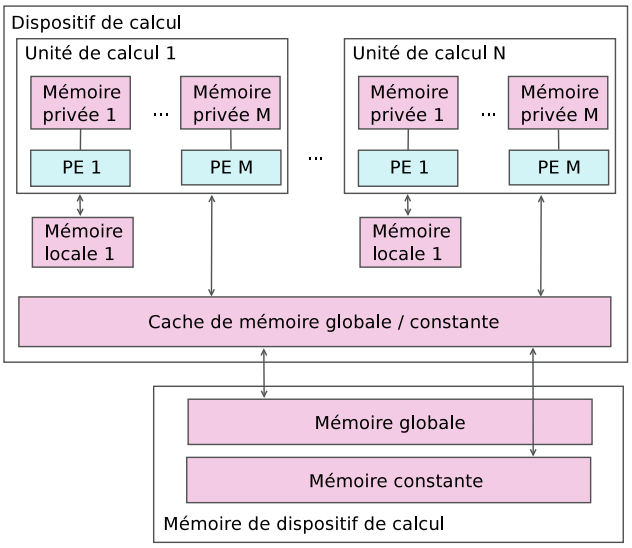


Figure 2.6 Architecture du dispositif OpenCL (voir Munshi, 2009, p. 25)

Le Tableau 2.1 montre quelques comparaisons entre les termes de CUDA et d'OpenCL.

Tableau 2.1 Comparaison des termes de modèle de mémoire

CUDA C	OpenCL
Registre	Mémoire privée
Mémoire locale	– *
Mémoire partagée	Mémoire locale
Mémoire de texture	Mémoire globale
Mémoire constante	Mémoire constante

* Mémoire locale de CUDA C est un peu comme mémoire privée d'OpenCL, mais celle de CUDA réside dans la mémoire globale

2.2.3 Notes sur OpenCL

Nous avons trouvé que l'accès à l'image par les fonctions clCreateImage2D et read_imageui est un peu plus lent que l'accès par la fonction clCreateBuffer et par une chaîne de caractère de 2 dimensions.

Une structure de boucle peut ralentir l'exécution, même si c'est juste une structure qui s'exécute une seule fois.

L'implémentation de compilation de virgule flottante de GPGPU de NVIDIA est un peu différente de celle de CPU d'Intel. Les résultats des exécutions avec GPGPU et CPU peuvent varier un peu dans l'emploi de virgule flottante.

CHAPITRE 3

REVUE DE LITTÉRATURE

Les images stéréoscopiques sont prises avec des appareils photo spécifiques. Nous présentons quelques exemples d'appareils photo stéréoscopiques binoculaires dans la Figure 3.1. La plupart de ces exemples sont recensés par Mattoccia dans une présentation (voir Mattoccia, 2009, p. 6).

valdesystems.com

focusrobotics.com

visionst.com

videredesign.com

tyzx.com

ptgrey.com

nvela.com

minoru3dwebcam.com

kodak.com

fujifilm.com

sony.com

sony.com

Figure 3.1 Exemples d'appareils photo stéréoscopiques

Avant de parler de disparité de vision, il est important de modéliser les appareils photo qui sont les outils essentiels des études.

3.1 Modèle de sténopé

Un modèle de sténopé est largement employé dans le domaine de vision par ordinateur. Un sténopé est un dispositif optique sans lentille permettant d'obtenir des images photographiques sous la forme d'une chambre noire. Il s'agit d'une ouverture de très faible diamètre. On appelle ainsi l'appareil photographique utilisant un tel dispositif. Nous avons présenté un peu d'histoire dans l'Annexe A. Xu et Zhang ont présenté ce modèle comme la Figure 3.2.

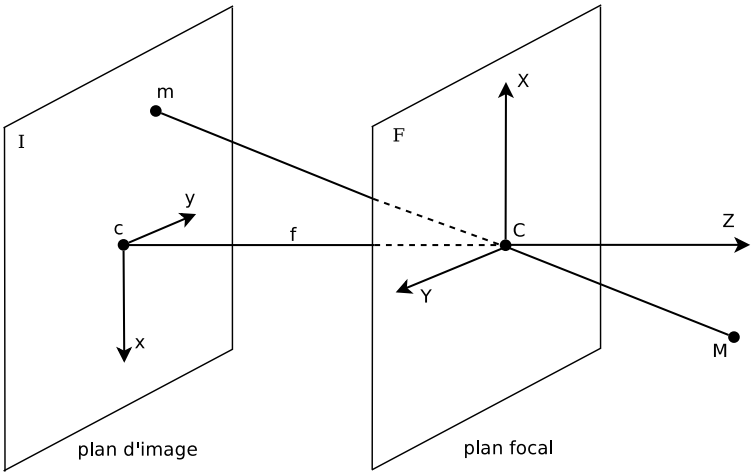


Figure 3.2 Modèle de sténopé (voir Xu et Zhang, 1996, p. 8)

Les lumières émises ou reflétées d'un objet passent par le trou de ce très faible diamètre pour former une image inverse de l'objet sur le plan d'image. La figure se compose d'un plan focal F et un plan d'image I. Le plan d'image est aussi appelé le plan de rétine. Le trou de très faible diamètre C se trouve dans le plan F. La ligne passante du centre optique C et perpendiculaire au plan d'image I est l'axe optique, qui s'entrecroise I au point c s'appelant le point principal. La relation entre les deux coordonnées 2D et 3D est exprimée par l'équation 3.1. (voir Xu et Zhang, 1996, pp. 7–9)

$$-\frac{x}{X} = -\frac{y}{Y} = \frac{f}{Z} \tag{3.1}$$

On peut ignorer la différence de direction si on choisit un autre système de coordonnées à la direction opposée de X et de Y. L'équation 3.1 peut être écrite comme l'équation 3.2 : (voir Xu et Zhang, 1996, p. 10)

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.2}$$

Étant donné un vecteur $\mathtt{x} = [x, y, \cdots]^T$, nous employons $\tilde{\mathtt{x}}$ pour désigner son vecteur augmenté en ajoutant 1 comme son dernier élément. (voir Xu et Zhang, 1996, p. 10)

Désignons la matrice de perspective par une matrice 3×4 :

$$P = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

L'équation 3.2 peut s'écrire comme (voir Xu et Zhang, 1996, p. 10)

$$s\tilde{m} = P\tilde{M}, \text{ où } s = S \text{ est un scalaire arbitraire non zéro} \tag{3.3}$$

3.2 Modélisation d'appareil photo et rectification d'images

Un modèle mathématique d'appareil photo décrit la transformation d'un point dans le système de coordonnées du monde vers un point dans une image. La transformation peut être divisée en deux : transformations extrinsèque et intrinsèque. La transformation extrinsèque décrit la relation entre le système de coordonnées du monde et celui du modèle des coordonnées d'un appareil photo. La transformation intrinsèque décrit la projection du système de coordonnées d'un appareil photo vers l'image. (voir Hirschmüller, 2003, p. 8)

3.2.1 Paramètres extrinsèques d'appareil photo

Si les points 3D sont dans un système de coordonnées autre que celui de l'appareil photo comme le cas dans la Figure 3.3, où un certain point 3D M représenté par $M_p = [X, Y, Z]^T$ et par $M_w = [X_w, Y_w, Z_w]^T$ respectivement dans les systèmes de coordonnées d'un appareil photo et du monde, et si son image est désignée par $m = [x, y]^T$, alors nous avons la relation ci-dessous dans l'équation 3.4. Nous avons également l'équation 3.5 pour décrire la relation entre l'objet 3D et son image. D est la matrice de paramètres extrinsèques de l'appareil photo. (voir Xu et Zhang, 1996, pp. 10–11)

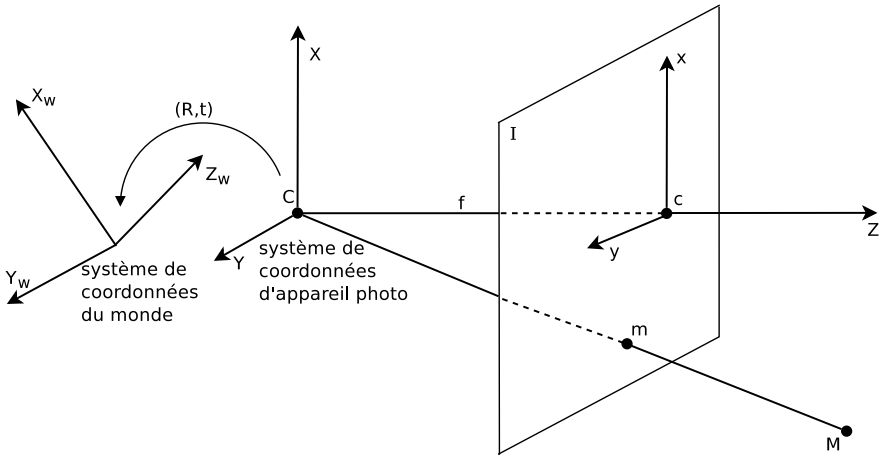


Figure 3.3 système de coordonnées du monde et les paramètres extrinsèques d'appareil photo (voir Xu et Zhang, 1996, p. 11)

$$\tilde{M}_p = D\tilde{M}_w, \text{où } D = \begin{bmatrix} R & t \\ 0_3^T & 1 \end{bmatrix} \text{ et } 0_3 = [0,0,0]^T \tag{3.4}$$

$$\tilde{m} = P\tilde{M}_p = PD\tilde{M}_w \tag{3.5}$$

3.2.2 Paramètres intrinsèques d'appareil photo

La Figure 3.4 montre un système de coordonnées (c, x, y) centré sur le point principal c et ayant les mêmes unités sur les axes x et y. Le système de coordonnées (o, u, v) nous permet d'accéder aux pixels dans une image. Supposons que $[u_0, v_0]^T$ désigne le point principal c dans (o, u, v), que k_u et k_v désignent les unités sur les axes u et v par rapport à celles employées dans le (c, x, y) et que θ désigne l'angle entre l'axe u et l'axe v, ces 5 paramètres ne dépendant pas de la position et de l'orientation de l'appareil photo, sont appelés les paramètres intrinsèques de l'appareil photo. (voir Xu et Zhang, 1996, p. 12–13)

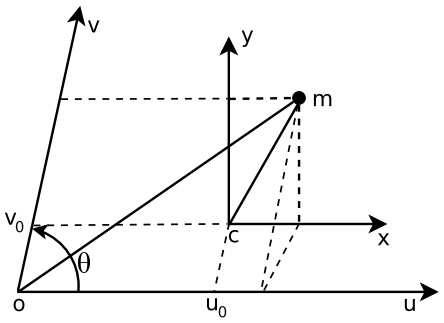


Figure 3.4 Les paramètres intrinsèques d'appareil photo (voir Xu et Zhang, 1996, p. 12)

Supposons que l'axe u et l'axe x sont parallèles, étant donné un point m, si $m_{\text{ancien}} = [x, y]^T$ représente les coordonnées dans le système de coordonnées original, si $m_{\text{nouveau}} = [u, v]^T$ représente les coordonnées dans le nouveau système de coordonnées, alors nous avons (voir Xu et Zhang, 1996, p. 13)

$$\tilde{m}_{\text{nouveau}} = H\tilde{m}_{\text{ancien}}, \text{ où } H = \begin{bmatrix} k_u & k_u \cot\theta & u_0 \\ 0 & k_v/\sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Selon l'équation 3.3, nous avons (voir Xu et Zhang, 1996, p. 13)

$$s\tilde{m}_{\text{ancien}} = P_{\text{ancien}}\tilde{M}$$

Alors

$$s\tilde{m}_{\text{nouveau}} = HP_{\text{nouveau}}\tilde{M}$$

Donc (voir Xu et Zhang, 1996, p. 13)

$$P_{\text{nouveau}} = HP_{\text{ancien}} = \begin{bmatrix} fk_u & fk_u \cot\theta & u_0 & 0 \\ 0 & fk_v/\sin\theta & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.7}$$

La matrice dépend des paramètres fk_u et fk_v. Un changement de longueur focale et un changement d'unité de pixel sont indiscernables. Nous pouvons les remplacer par deux paramètres $\alpha_u = fk_u$ et $\alpha_v = fk_v$. (voir Xu et Zhang, 1996, p. 13)

Nous pouvons définir un système de coordonnées normalisé d'un appareil photo où le plan d'image se trouve à une distance d'unité du centre optique. La matrice de perspective est

donnée par : (voir Xu et Zhang, 1996, p. 14)

$$P_N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Les coordonnées normalisées d'un point du monde $[X, Y, Z]^T$ dans le système de coordonnées d'un appareil photo sont : (voir Xu et Zhang, 1996, p. 14)

$$\begin{aligned} \hat{x} &= \frac{X}{Z} \\ \hat{y} &= \frac{Y}{Z} \end{aligned} \tag{3.8}$$

La matrice P définie par l'équation 3.7 peut être décomposée en un produit de deux matrices : (voir Xu et Zhang, 1996, p. 14)

$$P_{\text{nouveau}} = AP_N, \text{ où } A = \begin{bmatrix} \alpha_u & \alpha_u \cot\theta & u_0 \\ 0 & \alpha_v / \sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

La matrice A est appelée la matrice intrinsèque d'un appareil photo parce qu'elle ne contient que des paramètres intrinsèques. Les coordonnées d'image normalisées peuvent être données par l'équation 3.10. (voir Xu et Zhang, 1996, p. 14)

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} = A^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.10}$$

Ces transformations peuvent nous libérer des caractéristiques spécifiques d'un appareil photo et nous permettre de nous concentrer sur les problèmes essentiels. (voir Xu et Zhang, 1996, p. 14)

3.2.3 Calibration d'appareils photo

On emploie souvent un damier planaire pour calibrer un système d'appareils photo dans lequel on positionne le damier dans plusieurs inclinaisons pour permettre à un outil de rechercher des coins de grille. Plusieurs programmes gratuits sont disponibles tels OpenCV et un outil en MATLAB. (voir Bouguet, 2010).

3.2.4 Rectification d'images

La rectification d'images est le processus par lequel on projette les images étudiées sur un plan parallèle à la ligne de base rejoignant les deux centres optiques. (voir Forsyth et Ponce, 2002, p. 236) Horaud et Monga ont présenté en détail la rectification d'image (voir Horaud et Monga, 1995, p. 174–177). Trucco et Verri ont défini la rectification d'image comme une transformation de chaque image pour que chaque paire de lignes épipolaires conjuguées devienne colinéaire et parallèle à l'un des axes d'image, normalement l'axe horizontal. L'importance de la rectification est de réduire la recherche dans 2D vers celle 1D sur une ligne de balayage. (voir Trucco et Verri, 1998, p. 157)

La Figure 3.5 montre la rectification selon Trucco et Verri (voir Trucco et Verri, 1998, p. 159)

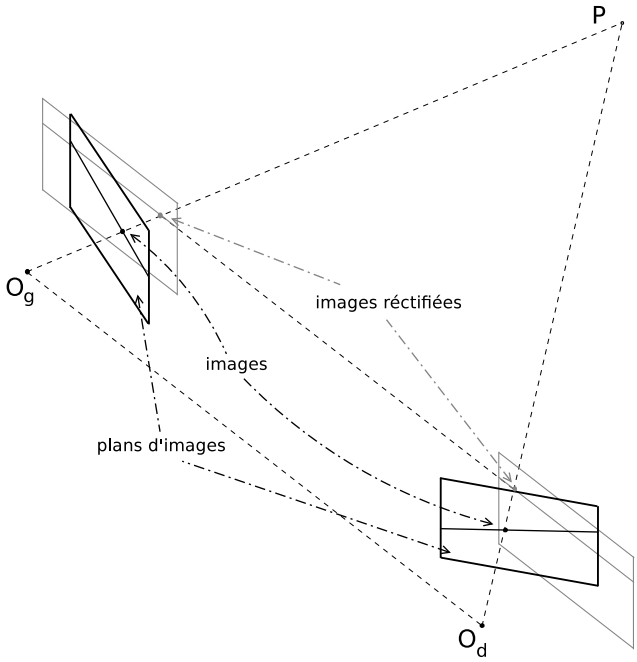


Figure 3.5 Rectification des deux images stéréoscopiques (voir Trucco et Verri, 1998, p. 159)

Pour faciliter le traitement, on doit avoir des images rectifiées. À partir de maintenant, nous supposons que nous avons des paires d'images rectifiées comme entrées.

3.3 Filtres

On emploie souvent de filtres pour éliminer de points indésirables afin de limiter l'influence provenant du bruit. Il existe plusieurs types de filtres concernant la stéréoscopie.

3.3.1 Filtre bilatéral

Pour lisser une image et en même temps conserver les bordures, Tomasi et Manduchi ont introduit un filtre qui d'une part renforce la situation à la fois géométrique et photométrique, et d'autre part remplace le pixel par une moyenne des pixels voisins similaires. On dit qu'il combine les filtres de domaine et de rang. Les filtres de domaine et de rang sont employés pour mesurer respectivement la similarité géométrique et photométrique.

Le filtre est décrit comme :

$$\begin{aligned} \mathbf{h}(\mathbf{x}) &= k^{-1}(\mathbf{x}) \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathbf{f}(\xi) c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x})) d\xi \\ \text{avec la normalisation } k(\mathbf{x}) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} c(\xi, \mathbf{x}) s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x})) d\xi \end{aligned} \tag{3.11}$$

où $c(\xi, \mathbf{x})$ mesure la proximité géométrique entre le pixel $\mathbf{x}$ et un point à proximité ξ, $s(\mathbf{f}(\xi), \mathbf{f}(\mathbf{x}))$ mesure la dissimilarité photométrique entre le pixel $\mathbf{x}$ et un point à proximité ξ. Ce filtre manifeste un bon comportement à la fois aux frontières et à la bordure, et il n'est pas itératif. (voir Tomasi et Manduchi, 1998, p. 840)

3.3.2 BilSub

Ansar *et al.* ont présenté une méthode stéréoscopique avec compensation de la variation photométrique entre les appareils photo de gauche et de droite, qui constitue une combinaison de corrélation par SDA et le filtre bilatéral de Tomasi et Manduchi (voir Tomasi et Manduchi, 1998). La compensation est normalement faite avec un filtre de Laplacien de Gaussienne (LoG), qui supprime le bruit de haute fréquence et en même temps normalise l'intensité en préservant la texture. Ce dernier peut être correctement approché par une Différence de Gaussiennes (DoG) :

$$I' = I * G(\sigma_{\text{petit}}) - I * G(\sigma_{\text{grand}}) \tag{3.12}$$

Pour des images de bonne qualité, un filtre qui élimine la haute fréquence n'est pas nécessaire. Donc, nous avons un filtre de soustraction de fond :

$$I' = I - I * G(\sigma_{\text{grand}}) \tag{3.13}$$

Ensuite, nous remplaçons la partie gaussienne $I * G(\sigma_{\text{grand}})$ par l'image filtrée bilatéralement B pour avoir un filtre

$$I' = I - B \tag{3.14}$$

Ce filtre peut avoir un très bon résultat par comparaison avec celui de corrélation croisée

normalisée. Il est utile pour éliminer les différences radiométriques. (voir Ansar *et al.*, 2004, pp. 2–4) Il va mieux d'employer BilSub dans l'espace de couleur CIELab comme suggérée par Tomasi et Manduchi. (voir Tomasi et Manduchi, 1998) Il y a des méthodes qui profitent l'avantage de rapidité de traitement de BilSub pour l'employer à filtrer les images avec différences radiométriques, par exemple, Matthies *et al.* ont présenté un système de navigation pour véhicules robotisés (voir Matthies *et al.*, 2007). Le travail de Halatci *et al.* a également employé des fonctions basées sur BilSub (voir Halatci *et al.*, 2007).

3.3.3 Filtre d'erreurs en provenance des fonctions de corrélation

Lorsque la valeur minimale et la seconde minimale sont notées comme C_1 et C_2, la position de correspondance ne sera pas certaine si C_1 et C_2 sont trop proches. La différence relative est :

$$C_d = \frac{C_2 - C_1}{C_1} \tag{3.15}$$

Si C_d est sous un seuil s_d, on rejette la disparité. Le seuil s_d dépend de l'application (voir Hirschmüller *et al.*, 2002, p. 235). Hirschmüller a indiqué que $C_2 - C_1 \geq s_d$ sera plus rapide à calculer et fonctionnera tout aussi bien. Évidemment, s_d dépend de l'application employée. (voir Hirschmüller, 2003, p. 32)

3.3.4 Filtre de correction de bordure

Il est important de préciser la bordure parce que souvent la vraie bordure se trouve quelques pixels ailleurs de celle calculée. Donc, une analyse plus détaillée concernant le gradient vertical sera demandée pour préciser les bordures (voir Hirschmüller, 2001). On suppose que la disparité reste constante dans une petite zone (voir Hirschmüller, 2003, p. 34). Pour une zone avec beaucoup de variation de disparité, il y aura un compromis entre la taille de bloc de correspondance et la précision de disparité.

Les filtres invalident des correspondances et créent des trous dans la carte de disparité. Il y a certaines applications qui demandent une distribution uniforme de disparités. Dans ce cas, on emploie l'interpolation.

3.4 Disparité

Marr a été le premier à parler de disparité de vision. Il a défini trois règles pour un algorithme de correspondance : compatibilité, unicité et continuité. La compatibilité fait référence à une même notation dans l'ensemble de l'étude. L'unicité veut dire qu'un point

peut correspondre à un seul point dans l'autre image. La continuité signifie que la disparité varie de façon régulière un peu partout. (voir Marr, 1982, p. 115)

En 2004, Isgro *et al.* ont classé les algorithmes de correspondance en deux catégories qui cherchent à obtenir un ensemble clairsemé de points de correspondance ou un ensemble dense de points de correspondance (voir Isgro *et al.*, 2004, p. 9).

Ensuite, Isgro *et al.* ont classé les algorithmes de disparité clairsemée en deux sous-catégories : correspondance par caractéristique et correspondance par gabarit. Les algorithmes dans la première sous-catégorie choisissent indépendamment des points de caractéristique dans les deux images, puis les assortissent en employant une séquence d'opérations : recherche par arbre, relaxation, détection de clique maximale et correspondance de chaîne ou approche algébrique basée sur les positions des points et les mesures de corrélation. Les algorithmes dans la deuxième sous-catégorie choisissent les gabarits dans une image puis cherchent les points correspondants dans l'autre image en employant une mesure de similarité. (voir Isgro *et al.*, 2004, p. 10)

La disparité dense produit une carte de disparité lisse. Les points de correspondance doivent satisfaire quelques contraintes comme : ordre, lissage et unicité. La contrainte de lissage est bien la continuité dont nous avons parlé plus haut. Les points sont généralement appariés par les méthodes de correspondance de type corrélation (voir Scharstein et Szeliski, 2002) : étant donné une fenêtre dans une image, les méthodes standard cherchent toutes les fenêtres possibles dans l'espace de l'autre image et choisissent celle qui optimise la métrique de similarité. Les métriques typiques incluent SDC (somme des différences au carré), SDA (somme des différences absolues) et corrélation. (voir Isgro *et al.*, 2004, p. 10)

Ce sujet a été très bien traité dans un article par Scharstein *et al.*, qui ont observé que la disparité dense est suscitée par les applications de stéréoscopie, comme la synthèse de vue et le rendu à base d'image. Ces applications ont besoin d'une évaluation de la disparité dans toutes les parties de l'image (voir Scharstein *et al.*, 2001, p. 1).

Dans cet article de Scharstein *et al.*, les auteurs supposent que les images sont prises dans un plan linéaire avec l'axe optique perpendiculaire au déplacement d'un appareil photo. La coordonnée (x, y) de l'espace de disparité est prise pour coïncider avec les coordonnées pixels d'une image de référence. La correspondance entre un pixel (x, y) dans l'image de référence et un pixel (x', y') dans l'image à faire correspondre est donnée par (voir Scharstein *et al.*, 2001, p. 3) :

$$\begin{cases} x' = x + s\, d(x, y) & \text{, où } s = \pm 1 \text{ pour que la disparité soit positive} \\ y' = y \end{cases} \tag{3.16}$$

C'est bien aussi l'hypothèse que nous avons utilisée dans nos études.

3.5 Mise en correspondance stéréoscopique

3.5.1 Contrôle de cohérence

Fua a présenté une méthode simple de contrôle de cohérence qui consiste à vérifier la cohérence de disparité en s'assurant de la compatibilité entre les deux directions gauche-droite / droite-gauche, permettant ainsi d'éviter la zone d'occlusion. Si les deux directions sont correspondantes, la disparité est cohérente. (voir Fua, 1991, p. 1293)

L'auteur a aussi énuméré des cas de malfonctionnement : (voir Fua, 1991, p. 1294)

- Les zones à corréler ont peu de texture
- La disparité varie rapidement dans la fenêtre de corrélation
- La présence d'une occlusion

Le contrôle de cohérence est souvent appelé la vérification de consistance dans nos études.

3.5.2 Mise en correspondance stéréoscopique en détail

Normalement, avant d'appliquer la corrélation, la méthode de Laplacien de Gaussienne ou de normalisation est utilisée pour compenser les différences de luminosité et de contraste. De plus, le contrôle de cohérence gauche-droite / droite-gauche introduit par Fua est largement utilisé. Une autre méthode nommée « opérateur d'intérêt (*Interest Operator*) » (voir Moravec, 1977) peut identifier des zones sans texture. Elle cherche les régions qui ont des éléments maximaux locaux sur une mesure de variance directionnelle (voir Moravec, 1977). La Méthode filtre de segment (voir Matthies *et al.*, 1995; Murray et Little, 2000) élimine des zones de disparité isolées. On peut aussi employer une interpolation quadratique sur les valeurs de meilleure corrélation pour arriver à une précision inférieure au pixel. (voir Hirschmüller, 2003, p. 22)

Faugeras *et al.* ont employé un système stéréoscopique trinoculaire dans lequel les appareils photo sont situés sur les sommets d'un triangle ayant un angle droit. L'image à l'angle droit est choisie comme référence pour être corrélée horizontalement avec une deuxième image et verticalement avec la dernière image. Les deux cartes de disparité sont ensuite fusionnées pour avoir une carte de disparité finale. (voir Faugeras *et al.*, 1993, pp. 9–10) Nous employons le système stéréoscopique binoculaire dans ces études.

La mise en correspondance peut se faire avec une fenêtre rectangulaire qui se déplace dans l'autre image. Le principal problème provient d'erreurs de correspondance. Le bruit dans les valeurs de pixel, l'ambiguïté de texture et la réflexion peuvent fortement influencer la correspondance. Une fenêtre plus grande peut diminuer l'effet de bruit, mais elle peut aussi

détériorer la qualité. Normalement, la taille de la fenêtre de corrélation est un compromis entre différentes erreurs de correspondance. La corrélation stéréoscopique brouille la forme des objets avec une tendance sous-jacente à étendre des objets horizontalement, et cet effet dépend de la taille de la fenêtre de corrélation. Parce que la similarité des zones de fond est normalement plus haute que celle d'objet, une grande fenêtre de corrélation peut faire correspondre le fond autour d'un objet et prendre l'objet comme bruit. (voir Hirschmüller, 2003, pp. 24, 26)

Hirschmüller a proposé un algorithme de stéréoscopie de fenêtres multiples et de filtres multiples. Les images sources rectifiées sont filtrées pour éliminer une polarisation constante de l'intensité due aux différences d'appareils photo. Ceci peut se faire avec un filtre moyen. Konolige a proposé d'utiliser la méthode de LoG (Laplacien de Gaussienne) qui peut aussi réduire le bruit par lissage gaussien. Une fenêtre rectangulaire définie par chaque pixel regroupant le voisinage du pixel est corrélée à toutes les fenêtres de la deuxième image, définie de la même façon, et ce, dans toutes les positions possibles. Cette opération est illustrée par la Figure 3.6. L'opération peut être optimisée en employant un calcul récursif suggéré par Faugeras *et al.*. Après, l'auteur a remplacé les valeurs de corrélation par une combinaison de valeurs voisines. Ensuite, un filtre d'erreur de corrélation de fonction est employé pour réduire des discordances générales. Le résultat sera passé par un contrôle de cohérence gauche-droite / droite-gauche puis par une interpolation inférieure au pixel. Finalement, on passe un filtre de correction de bordure et un filtre de segment pour mieux définir les bordures et réduire des petites zones isolées. Si nécessaire, des zones peuvent être interpolées encore. (voir Hirschmüller, 2003, pp. 28–29).

Les méthodes non paramétriques Rank et Census qui comparent les ordres d'intensité locale peuvent être à l'origine d'une perte d'information. Pour régler le problème de précision dans la zone des discontinuités de disparité, Hirschmüller a proposé une approche dite « approche de fenêtres multiples de support » qui additionne la valeur du pixel de centre et celles de quelques pixels voisins choisis. (voir Hirschmüller *et al.*, 2002, pp. 233–234)

Nous avons aussi des algorithmes qui ne désignent pas explicitement des fonctions globales qui doivent être minimisées, mais des comportements qui imitent étroitement celui d'algorithmes d'optimisation itérative.

3.5.3 Méthode récursive de Fugeras

On a besoin de connaître la probabilité de validité de correspondance. Une correspondance dans une zone dense de la carte de disparité a une valeur élevée de probabilité de corrélation, sauf pour des motifs répétitifs. La forme de la courbe de corrélation peut être employée pour évaluer la probabilité d'une correspondance correcte. Une courbe ayant plusieurs sommets

ou une large envergure peut augmenter la probabilité d'une mauvaise correspondance. (voir Faugeras *et al.*, 1993, p. 8)

La méthode de Faugeras *et al.* calcule le critère de corrélation de façon incrémentale. La fenêtre de corrélation a la dimension $(2n+1) \times (2m+1)$.

Des critères de corrélation peuvent être :

$$C_1(x,y,d) = \frac{\sum_{i,j} |I_1(x+i,y+j) - I_2(x+d+i,y+j)|^2}{\sqrt{\sum_{i,j} I_1(x+i,y+j)^2} \times \sqrt{\sum_{i,j} I_2(x+d+i,y+j)^2}}$$

$$C_2(x,y,d) = \frac{\sum_{i,j} I_1(x+i,y+j) \times I_2(x+d+i,y+j)}{\sqrt{\sum_{i,j} I_1(x+i,y+j)^2} \times \sqrt{\sum_{i,j} I_2(x+d+i,y+j)^2}}$$

$$C_3(x,y,d) = \frac{\sum_{i,j} |(I_1(x+i,y+j) - \overline{I_1(x,y)}) - (I_2(x+d+i,y+j) - \overline{I_2(x+d,y)})|^2}{\sqrt{\sum_{i,j} |I_1(x+i,y+j) - \overline{I_1(x,y)}|^2} \times \sqrt{\sum_{i,j} |I_2(x+d+i,y+j) - \overline{I_2(x+d,y)}|^2}}$$

$$C_4(x,y,d) = \frac{\sum_{i,j} |I_1(x+i,y+j) - \overline{I_1(x,y)}| \times |I_2(x+d+i,y+j) - \overline{I_2(x+d,y)}|}{\sqrt{\sum_{i,j} |I_1(x+i,y+j) - \overline{I_1(x,y)}|^2} \times \sqrt{\sum_{i,j} |I_2(x+d+i,y+j) - \overline{I_2(x+d,y)}|^2}}$$

Nous pouvons voir que la valeur de C_1 ne change pas si I_1 et I_2 sont remplacées par aI_1+b et aI_2+b, que la valeur de C_2 ne change pas si I_1 et I_2 sont remplacées par aI_1 et aI_2, que la valeur de C_3 ne change pas si I_1 et I_2 sont remplacées par aI_1+b_1 et aI_2+b_2, et que la valeur de C_4 ne change pas si I_1 et I_2 sont remplacées par $a_1I_1+b_1$ et $a_2I_2+b_2$. C_3 et C_4 ont une meilleure performance parce qu'elles sont moins affectées par les applications affines d'images qui peuvent être le résultat de petites différences de paramètres d'appareils photo.

C_2 a une performance similaire à celle de C_3 et C_4, sauf quand la différence de distribution de niveaux gris entre les images est importante. (voir Faugeras *et al.*, 1993, pp. 6–7)

Algorithme d'accélération de calcul

Prenons l'exemple de C_2. Nous avons observé que le premier terme dans le dénominateur est constant tant que x et y ne varient pas. Donc, le critère peut être simplifié à :

$$C_2'(x,y,d) = \frac{\sum_{i,j} I_1(x+i,y+j) \times I_2(x+d+i,y+j)}{\sqrt{\sum_{i,j} I_2(x+d+i,y+j)^2}} \tag{3.17}$$

Cette équation 3.17 peut être divisée en plusieurs parties :

$$\begin{aligned} O(x,y) &= \max_d \{N(x,y,d) \times R(x+d,y)\} \\ N(x,y,d) &= \sum_{i,j} I_1(x+i,y+j) \times I_2(x+d+i,y+j) \\ R(x,y) &= 1/\sqrt{M(x,y)} \\ M(x,y) &= \sum_{i,j} I_2(x+i,y+j) \times I_2(x+d+i,y+j) \end{aligned} \tag{3.18}$$

Le numérateur a $(2n+1) \times (2m+1)$ multiplications redondantes. Pour accélérer, nous calculons le numérateur N comme suit : (voir Faugeras *et al.*, 1993, pp. 10–11)

$$\begin{aligned} P(x,y,d) &= I_1(x,y) \times I_2(x+d,y) \\ Q(x,0,d) &= \sum_j P(x,j,d) \\ Q(x,y+1,d) &= Q(x,y,d) + P(x,y+2m+1,d) - P(x,y,d) \\ N(0,y,d) &= \sum_i Q(0,y,d) \\ N(x+1,y,d) &= N(x,y,d) + Q(x+2n+1,y,d) - Q(x,y,d) \end{aligned} \tag{3.19}$$

De façon comparable, le dénominateur M sera comme suit :

$$\begin{aligned} P_2(x,y) &= I_2(x,y) \times I_2(x,y) \\ Q_2(x,0) &= \sum_j P_2(x,j) \\ Q_2(x,y+1) &= Q_2(x,y) + P_2(x,y+2m+1) - P_2(x,y) \\ M(0,y) &= \sum_i Q_2(0,y) \\ M(x+1,y) &= M(x,y) + Q_2(x+2n+1,y) - Q_2(x,y) \end{aligned} \tag{3.20}$$

Ce type d'optimisation est très efficace (voir Faugeras *et al.*, 1993, pp. 10–11). Nous verrons plus tard que nous avons employé une méthode similaire dans l'accélération de calcul de disparité par GPGPU.

3.6 Mise en correspondance par bloc (BM)

Nous pouvons calculer une carte de disparité par la correspondance d'intensité des deux images. Toutefois, sachant qu'il y a e souvent beaucoup de répétitions d'intensités sur une ou

plusieurs zones, la mise en correspondance par pixel n'est pas fiable. Pour régler ce problème, la méthode la plus souvent employée est d'ajouter d'information du voisinage pour augmenter la robustesse. C'est bien la méthode de base que nous avons employée dans ce mémoire.

La Figure 3.6 montre comment la mise en correspondance par bloc fonctionne. On prend une fenêtre pour additionner les différences tirées de comparaisons d'intensité afin que davantage d'information entre en compte dans le calcul de correspondance, toutefois, cela signifie à faire beaucoup d'additions Si c'est un produit quadratique, la demande de temps sera encore augmentée. Compte tenu la nature de la mise en correspondance par bloc, beaucoup de calcul d'accumulation de différences prend du temps, si la taille d'images est grande, il sera difficile de traiter les images en temps voulu.

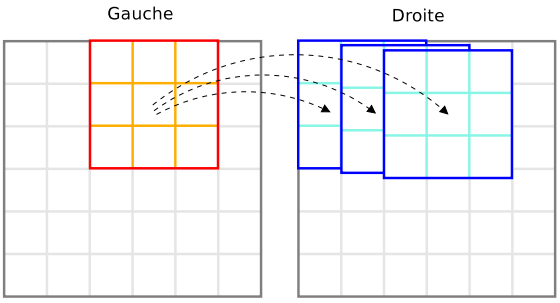


Figure 3.6 Calcul de disparité avec deux images

Le calcul de coût de mise en correspondance par bloc pose des problèmes :

- Coût élevé de calcul : l'accumulation de coût peut souvent ralentir l'exécution. Si le coût est un produit quadratique de différence, la demande de calcul sera encore élevée.
- Problème de texture légère : la disparité au moyen de la mise en correspondance par bloc souffre de trous (*dropout*) dans la zone ayant une texture légère. Konolige a présenté une méthode de projection de motif sur les objets pour les aider à avoir de texture afin de mieux faire la mise en correspondance. La méthode emploie de motif bien calculé afin d'aider la mise en correspondance par bloc. Konolige a présenté plusieurs aspects de la projection de motif, montré que pour calculer un motif, la méthode de distance minimale de Hamming est meilleure que celle de De Bruijn (voir Lim, 2009) et celle aléatoire. (voir Konolige, 2010)

3.7 Méthodes de calcul de coût ou méthode de correspondance

La mise en correspondance se fait avec une mesure de similarité,

3.7.1 Somme des différences au carré (SDC) et somme des différences absolues (SDA)

Souvent on emploie la somme des différences au carré (SDC) ou la somme des différences absolues (SDA) pour la mise en correspondance par bloc. La SDC est normalement meilleure que la SDA tandis que la SDA consume moins de ressource de calcul. On peut trouver beaucoup de méthodes basées sur ces deux mesures de différences.

3.7.2 Mesure de différence d'intensités avec la méthode d'interpolation

Birchfield et Tomasi ont indiqué que la différence d'intensité est grande à proximité des bordures. Parce que les solutions existantes ont toutes des contraintes, Birchfield et Tomasi ont proposé d'employer des fonctions d'interpolation linéaires au voisinage des deux pixels pour mesurer la dissemblance.

Si $\hat{I}_R$ est une fonction d'interpolation linéaire entre les points d'échantillonnage, la quantité est définie comme :

$$\bar{d}(x_i, y_i, I_L, I_R) = \min_{y_i - \frac{1}{2} \leq y \leq y_i + \frac{1}{2}} |I_L(x_i) - \hat{I}_R(y)| \tag{3.21}$$

La dissemblance entre les pixels est calculée comme le minimum de la quantité $\bar{d}(x_i, y_i, I_L, I_R)$ définie par l'équation 3.21 et sa contrepartie symétrique :

$$d(x_i, y_i) = \min\{\bar{d}(x_i, y_i, I_L, I_R), \bar{d}(y_i, x_i, I_R, I_L)\} \tag{3.22}$$

Pour calculer $\bar{d}(x_i, y_i, I_L, I_R)$, on calcule $I_R^- \equiv \hat{I}_R(y_i - \frac{1}{2}) = \frac{1}{2}(I_R(y_i) + I_R(y_i - 1))$, qui est l'intensité interpolée linéairement à mi-chemin entre y_i et son voisin de gauche. Et de manière similaire $I_R^+ \equiv \hat{I}_R(y_i + \frac{1}{2}) = \frac{1}{2}(I_R(y_i) + I_R(y_i + 1))$. En prenant $I_{\min} = \min(I_R^-, I_R^+, I_R(y_i))$ et $I_{\max} = \max(I_R^-, I_R^+, I_R(y_i))$, nous avons obtenu finalement :

$$\bar{d}(x_i, y_i, I_L, I_R) = \max\{0, I_L(x_i) - I_{\max}, I_{\min} - I_L(x_i)\} \tag{3.23}$$

(voir Birchfield et Tomasi, 1998, pp. 2–3)

3.7.3 Corrélation croisée normalisée (CCN)

Sun et Donate *et al.* ont employé la corrélation croisée normalisée (CCN) pour faire la mise en correspondance de fenêtres de comparaison, qui est définie par l'équation 3.24 (voir

Sun, 2002, p. 3) (voir Donate *et al.*, 2011, p. 186) :

$$C(i,j,d) = \frac{\text{cov}_{ij,d}(f,g)}{\sqrt{\text{var}_{ij}(f)} \times \sqrt{\text{var}_{ij,d}(g)}} \tag{3.24}$$

où (i,j) est les indices de ligne et de colonne d'un plan dans un certain volume de corrélation, d est la valeur possible de disparité. $\text{var}_{ij}(f)$ fait référence à la variance dans une fenêtre centrée au (i,j) de l'image f, $\text{cov}_{ij,d}(f,g)$ est la covariance entre les fenêtres des images de gauche et de droite. (voir Donate *et al.*, 2011, p. 186)

3.8 Implémentation de stéréoscopie par BM d'OpenCV

OpenCV est une bibliothèque graphique libre spécialisée dans le traitement d'images en temps réel. Eruhimov a fourni comme exemple d'application un programme de stéréoscopie (voir Eruhimov, 2010). Il a employé un algorithme stéréoscopique de mise en correspondance par bloc (BM) similaire à l'algorithme proposé par Konolige (voir Konolige, 1997).

3.8.1 BM d'une implémentation en CPU d'OpenCV

L'algorithme de stéréoscopie au moyen de la mise en correspondance par bloc commence par un filtre Sobel horizontal (voir PRATT, 2001, p. 456). Le noyau d'opération de convolution est :

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \tag{3.25}$$

Dans ce programme, la carte de disparité a une taille plus petite que l'image. Pour deux images de taille identique, la carte de disparité est calculée de la façon suivante :

$$\begin{aligned} r &= \text{plancher}(\text{taille}_{\text{bloc}}/2) \\ x_{\min} &= \text{disparité}_{\text{maximale}} + r \\ x_{\max} &= \text{largeur}_{\text{image}} - r \\ y_{\min} &= r \\ y_{\max} &= \text{hauteur}_{\text{image}} - r \end{aligned} \tag{3.26}$$

D'abord, on définit un paramètre *preFilterCap* qui prend une valeur de 1 à 63 pour un filtre selon un seuil choisi. Ensuite, on emploie ce paramètre *preFilterCap* pour créer un tableau de valeurs Tab centré sur le seuil. Les valeurs pour les éléments éloignés du centre d'une distance plus grande que le *preFilterCap* auront deux fois la valeur du seuil. Ce tableau sera employé pour lisser la carte de disparité.

Ensuite on crée un tampon t, dans une fenêtre de la taille du bloc, on assigne la valeur du tableau Tab selon l'indice des pixels de la première rangée de l'image de gauche. Pour chaque pixel de la largeur de l'image, on ajoute à chaque élément du tampon t la différence entre les valeurs du tableau Tab, lesquelles valeurs sont obtenues en fonction des indices des deux pixels correspondants aux deux fenêtres adjacentes comme dans la Figure 3.7.

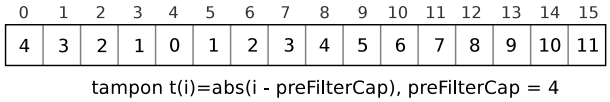


Figure 3.7 Valeurs du tampon t si $preFilterCap = 8$

On fait une somme de la première ligne des éléments de t, puis pour chaque ligne de l'image, on ajoute la différence entre les valeurs du tableau Tab telles que définies précédemment. Le résultat sera une somme des différences absolues des valeurs des deux fenêtres correspondantes. Pour s'assurer qu'il y a assez de texture dans la mise en correspondance afin de limiter l'influence du bruit aléatoire, on définit un seuil de texture, qui est une limite de réponse de fenêtre SDA (somme des différences absolues) pour que l'on ne considère pas les correspondances ayant une réponse sous le seuil. Bradski et Kaehler ont conseillé de s'assurer qu'il y a assez de texture pour contrer le bruit aléatoire pendant la mise en correspondance (voir Bradski et Kaehler, 2008, p. 443).

Si une valeur de disparité n'est pas valide, on suppose que le pixel se trouve dans une zone d'occlusion. Si une valeur de disparité existe, on va évaluer l'unicité de cette valeur selon un seuil d'unicité. Bradski et Kaehler ont observé que souvent la mise en correspondance a la caractéristique d'un fort pic central entouré de lobes secondaires (voir Bradski et Kaehler, 2008, p. 442). On peut employer cette observation pour filtrer de mauvaises correspondances.

Une fois que l'on a trouvé des valeurs possibles de disparité, on lisse encore une fois avec des valeurs autour du pixel étudié. En dernier lieu, on passe un filtre de tache pour éliminer de petites régions isolées qui sont considérées comme invalides. La mise en correspondance à base de blocs a des problèmes à proximité des limites des objets parce que la fenêtre de correspondance peut prendre le premier plan d'un côté et le fond de l'autre côté (voir Bradski et Kaehler, 2008, p. 443). Souvent ces problèmes correspondent à de petites régions séparées (*speckles*) dans la carte de disparité. Alors, le programme cherche ces régions pour en invalider les valeurs.

Le résultat obtenu avec cette méthode est bon visuellement, avec un taux d'erreur élevé toutefois. La Figure 3.8 montre deux cartes de disparité obtenues en employant les images de « *Art* » et de « *Laundry* » de taille de 463×370 de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005). Les taux d'erreur des pixels visibles sont 38% et 54% res-

pectivement selon notre calcul. La définition de taux d'erreur est présentée plus loin dans la sous-section 4.5.

BM, *Art*, taille : 5

BM, *Laundry*, taille : 5

Figure 3.8 Comparaison des cartes obtenues selon image

L'effet secondaire de cette méthode est que la carte de disparité peut être trop lissée. La performance dépend des paramètres employés et des images traitées. Pour améliorer la performance, ce programme supporte SSE2 (*Streaming SIMD Extensions 2*) pour accélérer le processus.

3.8.2 BM en GPGPU

Cet algorithme est implémenté sur GPGPU en CUDA C, qui traite 8 valeurs de disparités par chaque noyau pour profiter de la mémoire partagée. À part de ce point, l'algorithme est identique que celle nous allons présenter plus loin dans la sous-section 3.9. Il peut produire une carte de disparité de 463×370 avec la disparité maximale de 63 en environs 1 *ms* sur le NVIDIA GeForce GTX 580. Nous avons des chiffres de performance dans le Tableau 4.5 de la sous-section 4.10.6. Nous avons appliqué et amélioré cette méthode sur OpenCL.

3.9 Implémentation de la mise en correspondance par bloc en GPGPU

Stam a présenté un rapport de l'implémentation de la mise en correspondance par bloc en CUDA C (voir Stam, 2008). La Figure 3.9 (voir Stam, 2008, p. 10) illustre un schéma pour une fenêtre de comparaison de taille 3×3 et un bloc de fils d'exécution de taille 8×1. Dans l'application, un bloc d'au moins 32 fils d'exécution doit être employé. Un bloc de taille 64 ou 128 serait optimal.

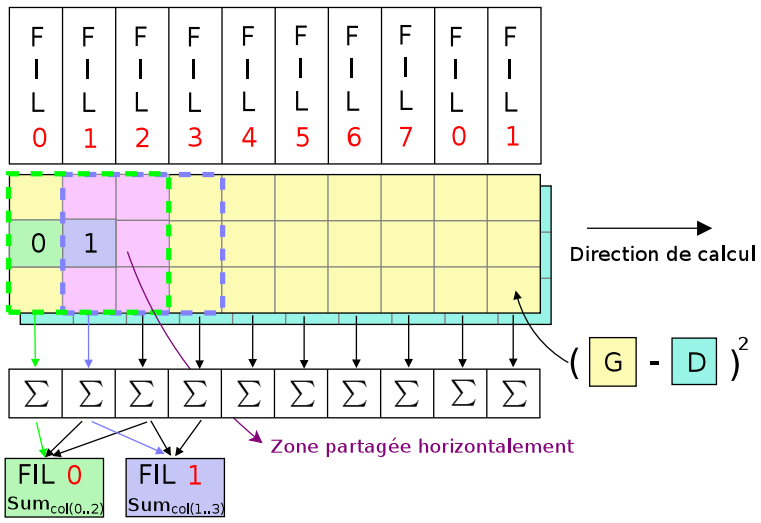


Figure 3.9 Fils d'exécution sur GPGPU (voir Stam, 2008, p. 10)

Chaque fil d'exécution accumule les différences quadratiques d'une colonne de pixels de la hauteur de la fenêtre de comparaison. La différence quadratique entre les pixels de l'image de gauche imgD et ceux de l'image de droite imgD est $\{\text{imgG}(x,y) - \text{imgD}(x+i,y)\}^2 (0 \leq i \leq d)$. La somme des écarts quadratiques de chaque colonne s_c est sauvegardée dans la mémoire locale. Parce qu'un pixel supplémentaire a été ajouté à chaque côté du bloc traité, deux fils d'exécution doivent faire une lecture et une somme supplémentaires (fils 0–1 dans la figure).

Quand on parle d'une ligne de pixels, chacun de ces pixels est au centre d'une fenêtre de comparaison horizontalement, mais aussi au centre de cette fenêtre verticalement. Pour la première ligne, nous devons calculer les écarts quadratiques afin d'obtenir s_c. Pour les lignes suivantes, puisqu'il n'y a que deux pixels différents entre la colonne de la ligne actuelle et la colonne de la ligne précédente, nous pouvons calculer la somme en soustrayant la première différence quadratique de chaque colonne de la ligne précédente puis en ajoutant la dernière différence quadratique de chaque colonne de la ligne actuelle. Voir la Figure 3.10 pour l'illustration.

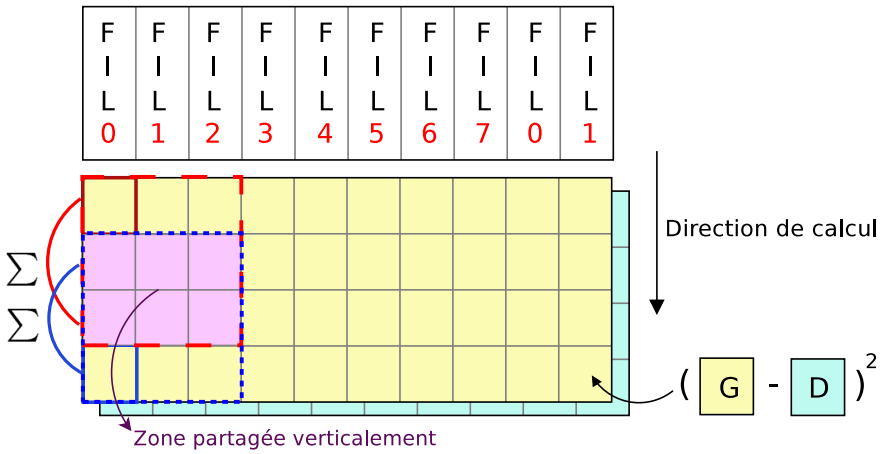


Figure 3.10 Fils d'exécution en colonne

La somme des s_c dans une fenêtre T_f est calculée pour trouver l'indice d_f de la valeur s_c minimale. Les indices d_f finaux produisent la carte de disparité recherchée.

3.10 Méthodes hiérarchiques

Les méthodes hiérarchiques peuvent accélérer le processus en guidant la recherche par les disparités calculées avec les images en basse résolution. Parce que la gamme de recherche est réduite, il est possible d'augmenter la précision. Ce type de méthodes peut être regroupé plus. Les algorithmes hiérarchiques ressemblent aux algorithmes itératifs, mais fonctionnent généralement sur une pyramide d'images, où les résultats de plus haut niveau (plus petite image) sont utilisés pour amener la recherche à un niveau plus détaillé (plus grande image). Par exemple, la méthode de propagation de croyance hiérarchique de Felzenszwalb et Huttenlocher (voir Felzenszwalb et Huttenlocher, 2004), aussi la méthode de propagation de croyance d'espace constant de Yang *et al.* (voir Yang *et al.*, 2010).

Il y a des méthodes hiérarchiques basées sur l'accumulation des coûts sur une plus grande de région, ou une réduction virtuelle de résolution sur qui on estime les disparités. Ce type

de méthodes hiérarchiques emploie normalement la propagation de croyance à chercher les disparités correspondant aux coûts minimaux.

3.10.1 Méthode hiérarchique basée sur la pyramide gaussienne

Ces méthodes hiérarchiques décomposent les paires d'images stéréoscopiques en une pyramide gaussienne. On estime une série de cartes de disparité de basse résolution vers une plus haute résolution en guidant la recherche basée sur les informations de carte de disparité de basse résolution. Les recherches seront limitées aux environs des disparités initialement identifiées. Cette approche permet d'augmenter la précision et la vitesse de traitement.

La définition de la pyramide gaussienne est :

$$\begin{aligned} g_0(x,y) &= I(x,y) \text{ pour le niveau de base,} I(x,y) \text{ est l'image originale} \\ g_l(x,y) &= \sum_{m=-2}^{2}\sum_{n=-2}^{2} w(m,n) g_{l-1}(2x+m, 2y+n) \text{ pour les autres niveaux,} \\ &\quad \text{et où } w(m,n) \text{ est une fonction de pondération} \end{aligned} \tag{3.27}$$

Si on enregistre les informations de différence d'image entre deux niveaux, on a la pyramide laplacienne.

Il y a des méthodes publiées basé sur la méthode hiérarchique avec la pyramide gaussienne, par exemple, celui de Lee et Sharma (voir Lee et Sharma, 2011).

Sizintsev *et al.* ont proposé une méthode qui permet de passer d'une vue d'ensemble imprécise à une vue de détail. On crée une pyramide d'images de la plus floue à la plus précise. On emploie une fenêtre non centrée pour faire une correspondance à chaque niveau de la pyramide. L'implémentation a employé CUDA C pour réaliser à 32 fps (images par seconde, ou *frames per second*) sur les vidéos de 640×480 avec une plage de recherche de disparité de 256. (voir Sizintsev *et al.*, 2010)

3.10.2 Correspondance basée sur le chemin

Donate *et al.* ont présenté un algorithme basé sur le chemin ayant une précision inférieure au pixel, cet algorithme lui-même est basé sur celui de Sun (voir Sun, 2002). L'algorithme emploie une pyramide d'images hiérarchique pour guider la mise en correspondance, une interpolation bilinéaire est employée pour remplir les pixels pour les points n'étant pas un multiple du ratio r qui est le ratio entre deux niveaux de résolution dans la pyramide d'images. (voir Donate *et al.*, 2011; Sun, 2002)

3.11 HMI et Census

Hirschmüller et Scharstein ont évalué des méthodes de calcul de coût de correspondance stéréoscopique qui tolèrent bien les différences radiométriques des images. Les meilleures méthodes de calcul de coût incluent BilSub (voir Ansar *et al.*, 2004, *BilSub*) qui fonctionne bien pour les différences radiométriques de basse fréquence, HMI qui est meilleure dans le cas de correspondance par pixel s'il y a un pourcentage élevé de bruit, et aussi Census qui affiche la meilleure et la plus robuste performance globale. (voir Hirschmüller et Scharstein, 2009)

3.11.1 HMI (information mutuelle hiérarchique, *hierarchical mutual information*)

Selon Hirschmüller, un algorithme local choisit une disparité avec le coût minimal de correspondance, tandis qu'un algorithme global ignore l'étape d'agrégation de coût et définit une fonction d'énergie globale qui inclut un terme de donnée et un terme de lissage (voir Hirschmüller, 2008, p. 328).

Hirschmüller a présenté la méthode SGM (*semi-global matching*), qui calcule les coûts de correspondance hiérarchiquement avec la méthode d'information mutuelle (*mutual information*) qui est définie de l'entropie H des deux images et de leur entropie jointe :

$$MI_{I_1,I_2} = H_{I_1} + H_{I_2} - H_{I_1,I_2} \tag{3.28}$$

L'entropie a été développée à partir du 18e siècle, propulsée par Shannon (voir Shannon, 1948), introduite dans la vision d'ordinateur par Viola et Wells (voir Viola et Wells, 1995) et améliorée dans le calcul de coût de donnée par Kim *et al.* (voir Kim *et al.*, 2003). Les entropies sont calculées de distributions de probabilité P des intensités des images associées (voir Hirschmüller, 2008, p. 329) :

$$H_I = -\int_0^1 P_I(i) \log P_I(i) di, \tag{3.29}$$

$$H_{I_1,I_2} = -\int_0^1 \int_0^1 P_{I_1,I_2}(i_1, i_2) \log P_{I_1,I_2}(i_1, i_2) di_1 di_2. \tag{3.30}$$

Kim *et al.* ont transformé le calcul de l'entropie jointe H_{I_1,I_2} en une somme sur les pixels à l'aide du développement de Taylor (voir Kim *et al.*, 2003). Donc l'entropie jointe est calculée comme une somme des termes de données qui dépendent des intensités correspondantes du

pixel p (voir Hirschmüller, 2008, p. 329) :

$$H_{I_1,I_2} = \sum_p h_{I_1,I_2}(I_{1p}, I_{2p}) \tag{3.31}$$

La distribution de probabilité des intensités correspondantes est définie par l'opérateur $T[]$ qui prend 1 si l'argument est vrai et 0 pour d'autres cas. (voir Hirschmüller, 2008, p. 329) :

$$P_{I_1,I_2}(i,k) = \frac{1}{n} \sum_p T[(i,k) = (I_{1p}, I_{2p})] \tag{3.32}$$

Hirschmüller emploie un calcul hiérarchique de carte de disparité pour guider le calcul de coût de la méthode d'information mutuelle. (voir Hirschmüller, 2008, p. 330)

Le terme de donnée h_{I_1,I_2} est calculé à partir de la distribution de probabilité jointe P_{I_1,I_2} des intensités correspondantes avec n comme le nombre de pixels. Le terme est calculé avec une convolution pour l'estimation Parzen (voir Parzen, 1962) qu'on peut trouver d'explications détaillées dans un livre de Duda et Hart (voir Duda et Hart, 1973).

$$h_{I_1,I_2} = -\frac{1}{n} \log(P_{I_1,I_2}(i,k) \otimes g(i,k)) \otimes g(i,k) \tag{3.33}$$

3.11.2 Census

Zabih et Woodfill ont introduit deux transformations non paramétriques locales : transformation Rank et transformation Census. La transformation Rank est une mesure non paramétrique d'intensité locale. La transformation Census est un sommaire non paramétrique de structure spatiale locale. (voir Zabih et Woodfill, 1994, p. 2)

Si P est un pixel, $I(P)$ est son intensité (normalement un entier encodé en 8-bit), et $N(P)$ un ensemble de pixels dans un voisinage correspondant à un carré de diamètre d entourant P. Définissons

$$\xi(P, P') = \begin{cases} 1 & \text{si } I(P') < I(P) \\ 0 & \text{sinon} \end{cases} \tag{3.34}$$

La transformation non paramétrique locale ne repose que sur l'ensemble de comparaisons de pixels qui est un ensemble de paires ordonnées :

$$\Xi(P) = \bigcup_{P' \in N(P)} (P', \xi(P, P')) \tag{3.35}$$

La transformation Rank $R(P)$ est :

$$R(P) = ||\{P' \in N(P) | I(P') < I(P)\}|| \tag{3.36}$$

La transformation Census correspond au voisinage entourant un pixel P à une chaîne de bits représentant l'ensemble de pixels voisins dont l'intensité est moins que celle de P. Prenons $N(P) = P \oplus D$, où $\oplus$ est la somme de Minkowski et D est un ensemble de déplacements, et laissons $\otimes$ représenter la concaténation. La transformation Census peut être spécifiée comme :

$$R_\tau = \bigotimes_{[i,j] \in D} \xi(P, P + [i, j]) \tag{3.37}$$

Deux pixels d'images de Census transformées sont comparés pour étudier leur similarité en utilisant la distance de Hamming, c'est-à-dire le nombre de bits qui diffèrent dans deux chaînes de bits. La correspondance est calculée comme la distance minimale de Hamming après l'application de la transformation Census. La valeur après la transformation repose plutôt sur le nombre de comparaisons d'intensité des pixels plutôt que sur les valeurs d'intensité (voir Zabih et Woodfill, 1994, pp. 2–3).

Le nombre de pixels dans la comparaison peut varier de 8 à 64. Plus le nombre de pixels (soit la taille de la fenêtre de comparaison) est grand, plus y auront d'informations pouvant être prises en compte, mais l'effet aux discontinuités sera aggravé. (voir Woodfill et Von Herzen, 1997, p. 206)

La distance de Hamming est calculée avec l'algorithme de Wegner, qui calcule bit à bit avec l'opérateur logique OU exclusif (XOR) sur deux entrées, puis continuellement trouve et efface le bit non zéro d'ordre le plus faible. Le temps d'exécution est proportionnel à la distance Hamming et non au nombre de bits dans les entrées. Nous avons le code en C comme suit. (voir Wegner, 2011)

```
unsigned hamdist(unsigned x, unsigned y){
   unsigned dist = 0, val = x ^ y;
   while(val){
      ++dist;
      val &= val - 1;
   }
   return dist;
}
```

3.11.3 Méthodes basées sur Census

Humenberger *et al.* ont introduit une méthode basée sur la segmentation en employant Census pour le calcul de coût local. La fiabilité de correspondance a été augmentée par une modification de correspondance semi-globale (SGM) introduite par Hirschmüller (voir Hirschmüller, 2008). La segmentation est appliquée sur l'image de texture binairement ou l'image stéréoscopique de gauche selon couleurs avec la méthode de *Mean Shift* (voir Fukunaga et Hostetler, 1975; Comaniciu et Meer, 2002). Le résultat intermédiaire de cette étape sera une carte de disparité initiale. Ensuite, on procède une segmentation sur l'image stéréoscopique de gauche ou la carte de texture afin d'appliquer un modèle planaire sur chaque segment.

Il est intéressant de noter quelques définitions. La fiabilité de correspondance a été définie comme suit :

$$CM(u,v) := \min\{255, 1024\big(\frac{\Delta(u,v)}{\text{MaxCosts}}\big)\} \tag{3.38}$$

où $\Delta(u,v)$ est la différence entre les deux meilleurs candidats de correspondance pour le pixel (u,v).

Il est difficile de faire correspondance dans une grande région sans texture. Pour commencer, on peut estimer une carte de texture en employant le filtre de variance sur une fenêtre de $n \times m$ avec :

$$TM(u,v) := \frac{1}{nm}\sum_{i=-n'}^{n'}\sum_{j=-m'}^{m'} I(u+i,v+j)^2 - \left(\frac{1}{nm}\sum_{i=-n'}^{n'}\sum_{j=-m'}^{m'} I(u+i,v+j)\right)^2 \tag{3.39}$$

On peut distinguer les pixels sans assez de confiance et les pixels dans les régions sans texture par l'emploi de deux seuils τ_1 et τ_2. Notons DM comme la carte de disparité estimée. La carte de disparité initiale est définie comme suit :

$$DM_{init}(u,v) := \begin{cases} DM(u,v) & \text{si } CM(u,v) \geq \tau_1 \wedge TM(u,v) \geq \tau_2 \\ 0 & \text{sinon} \end{cases} \tag{3.40}$$

Les auteurs ont ensuite montré l'application de la méthode de segmentation et de montage de plan sur les pixels sans texture et peu fiables pour une meilleure qualité de disparité. La segmentation peut se faire par couleur sur l'image d'entrée de gauche (*Mean Shift* (voir Fukunaga et Hostetler, 1975; Comaniciu et Meer, 2002)) ou binaire sur l'image de texture. L'image de texture (IT) est obtenue à partir de la carte de texture par :

$$IT(u,v) := \begin{cases} 0 & \text{si } TM(u,v) \leq t_{\text{texture}} \\ 255 & \text{sinon} \end{cases} \tag{3.41}$$

où t_{texture} est un seuil employé. Le processus de segmentation sur l'image de texture binaire est simple. Tous les pixels blancs sont unifiés dans un segment et tous les pixels noirs connectés sont mis dans un seul segment. (voir Humenberger *et al.*, 2010, p. 79)

Les pixels ayant passé la vérification de fiabilité sont employés à l'étape de montage de plan. Un plan est représenté par une équation de trois paramètres :

$$d(u, v) := au + bv + c. \tag{3.42}$$

On peut estimer les paramètres avec la méthode des moindres carrés. Toutefois, pour la robustesse, les auteurs ont employé une version robuste de Bleyer et Gelautz (voir Bleyer et Gelautz, 2005). La création de la carte de disparité finale nécessite une optimisation et un peaufinage de la carte initiale comme la dernière étape. (voir Humenberger *et al.*, 2010, pp. 79–80)

Weber *et al.* ont présenté une méthode d'accélération de Census en employant CUDA C sur GPGPU. Ils ont aussi indiqué que la vraie valeur de disparité se trouve quelque part entre les pixels en partie parce que l'on emploie un nombre entier dans le calcul. Donc, il faut alors prendre une valeur minimale de disparité plus des valeurs voisines pour en augmenter la précision en employant la méthode de montage parabolique. (voir Weber *et al.*, 2009, p. 790)

Zinner *et al.* ont présenté une implémentation rapide de l'algorithme Census sous forme d'un logiciel optimisé avec les instructions SSE (*Streaming SIMD Extensions*) et OpenMP. Les auteurs ont présenté une méthode de calcul rapide de distance Hamming et une amélioration de la performance de Census en en faisant un usage sélectif. (voir Zinner *et al.*, 2008, pp. 221–223)

Mei *et al.* ont présenté une combinaison de méthodes basées sur la différence absolue et la transformation Census. Pour un pixel $\mathbf{p} = (x, y)$ de l'image de gauche, la disparité d et le pixel correspondant $\mathbf{pd} = (x - d, y)$ de l'image de droite, on calcule deux valeurs de coût $C_{\text{Census}}(\mathbf{p}, d)$ et $C_{\text{AD}}(\mathbf{p}, d)$. (voir Mei *et al.*, 2011, p. 2)

$$C_{\text{Census}}(\mathbf{p}, d) = \text{distanceHamming}(I^{\text{gauche}}(\mathbf{p}), I^{\text{droite}}(\mathbf{pd})) \tag{3.43}$$

$$C_{\text{AD}}(\mathbf{p}, d) = \frac{1}{3} \sum_{i=R,V,B} |I_i^{\text{gauche}}(\mathbf{p}) - I_i^{\text{droite}}(\mathbf{pd})| \tag{3.44}$$

$$C(\mathbf{p}, d) = \rho(C_{\text{Census}}(\mathbf{p}, d), \lambda_{\text{Census}}) + \rho(C_{\text{AD}}(\mathbf{p}, d), \lambda_{\text{AD}}), \text{ où } \rho(c, \lambda) = 1 - \exp(-\frac{c}{\lambda}) \tag{3.45}$$

Ensuite, on emploie l'agrégation des coûts croisés (voir Zhang *et al.*, 2009), l'optimisation

de la ligne de balayage et le raffinement des disparités en multiples étapes pour avoir un bon résultat.

3.12 Méthodes d'interférence de SGM (*semi-global matching*)

Dans la méthode de SGM, l'agrégation des coûts est effectuée comme approximation d'une fonction globale d'énergie par les optimisations trajectorielle de toutes les directions à travers l'image comme illustrée dans la Figure 3.11 pour le cas de 16 directions. (voir Hirschmüller, 2008, p. 329)

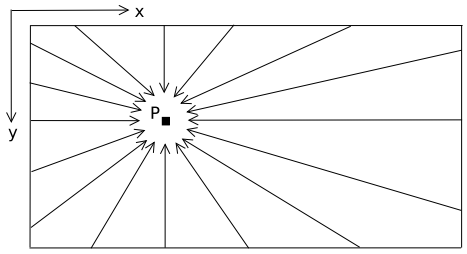


Figure 3.11 Agrégation des coûts dans un espace de disparité (voir Hirschmüller, 2008, p. 331)

Le coût $L_r(p,d)$ sur un chemin traversé dans la direction $\mathbf{r}$ du pixel $\mathbf{p}$ de la disparité d correspond à : (voir Hirschmüller, 2008, p. 331)

$$\begin{aligned} L_r(\mathbf{p},d) =& C(\mathbf{p},d) + \min\{L_r(\mathbf{p}-\mathbf{r},d),\ L_r(\mathbf{p}-\mathbf{r},d-1)+P_1, \\ & L_r(\mathbf{p}-\mathbf{r},d+1)+P_1,\ \min_i L_r(\mathbf{p}-\mathbf{r},i)+P_2\} \\ & - \min_k L_r(\mathbf{p}-\mathbf{r},k). \end{aligned} \tag{3.46}$$

$C(\mathbf{p},d)$ est une fonction de coût de correspondance par pixel, qui peut prendre la forme de fonction de HMI (information mutuelle hiérarchique) ou de la mesure d'intensité de Birchfield et Tomasi introduite dans la sous-section 3.7.2. $\min_k L_r(\mathbf{p}-\mathbf{r},k)$ est le coût minimal sur chemin du pixel précédent, on la soustrait pour s'assurer que L a une limite supérieure $L \leq C_{\max} + P_2$.

La carte de disparité correspond le coût minimal de chaque pixel par la programmation dynamique. Pour améliorer la qualité, différentes méthodes sont appliquées par la suite.

OpenCV a une implémentation du SGM présenté par Hirschmüller. Les coûts sont calculés avec l'algorithme de Birchfield et Tomasi. Ils sont ensuite additionnés selon une fenêtre de traitement. Comme dans le cas de la mise en correspondance par bloc, on effectue vers à fin du calcul une vérification d'unicité en fonction d'un seuil déterminé, puis on procède à une interpolation quadratique.

À la fin, on passe un filtre pour brouiller les bordures et un autre filtre de tache pour enlever de petits points isolés.

Si on utilise la programmation dynamique, on recourt à la méthode HH d'OpenCV, qui a normalement le meilleur résultat parmi les trois méthodes ici étudiées dans OpenCV.

3.13 Méthodes probabilistes

Compte tenu la difficulté dans la décision d'une correspondance, les méthodes probabilistes sont devenues les choix naturels. Nous n'en listons que quelques une ici.

3.13.1 Méthode de propagation de croyance

Comme la méthode de propagation de croyance demande beaucoup d'espace pour la sauvegarde de messages, la méthode utilisée est généralement limitée aux images à basse définition. Yang *et al.* ont proposé un algorithme qui demande un espace constant de mémoire. Cet algorithme peut s'opérer en temps linéaire, et ce, en fonction du nombre de pixels contenus dans l'image. La demande de mémoire est indépendante du nombre de niveaux de disparité L. Le temps d'exécution est aussi indépendant de L si on exclut le temps de calcul des données. (voir Yang *et al.*, 2010).

3.13.2 Approche probabiliste des images en haute définition

Geiger *et al.* ont présenté un travail intéressant. Avec une approche de point de support et probabiliste, ils ont obtenu une très bonne performance sur les images en haute définition avec CPU. Les points de support sont les pixels ayant une correspondance robuste grâce à leur texture et unicité, obtenus par la concaténation des réponses horizontales et verticales de filtre Sobel d'une fenêtre de 9×9. Les points de support sont ensuite employés pour créer une maille 2D avec l'algorithme de triangulation de Delaunay. Une distribution préliminaire est calculée à l'aide des disparités des points de support et la maille triangulée. (voir Geiger *et al.*, 2010)

Prenons $S = \{s_1, \ldots, s_M\}$ comme un ensemble de points de support. $O = \{o_1, \ldots, o_N\}$ est un ensemble d'observations d'image. $o_n^{(g)}$ et $o_n^{(d)}$ désignent les observations dans les images de gauche et de droite respectivement. (voir Geiger *et al.*, 2010, p. 5)

Supposons que les observations $\{o_n^{(g)}, o_n^{(d)}\}$ et les points de support S sont conditionnellement indépendants étant donné ses disparités d_n, la distribution jointe peut être factorisée à : (voir Geiger *et al.*, 2010, p. 5)

$$p(d_n, o_n^{(g)}, o_n^{(d)}, S) \propto p(d_n|S, o_n^{(g)})\, p(o_n^{(d)}|o_n^{(g)}, d_n) \tag{3.47}$$

$p(d_n|S, o_n^{(g)})$ est une distribution préliminaire et $p(o_n^{(d)}|o_n^{(g)}, d_n)$ est comme vraisemblance d'image. La Figure 3.12 est un modèle graphique décrit par l'équation 3.47. (voir Geiger *et al.*, 2010, p. 5)

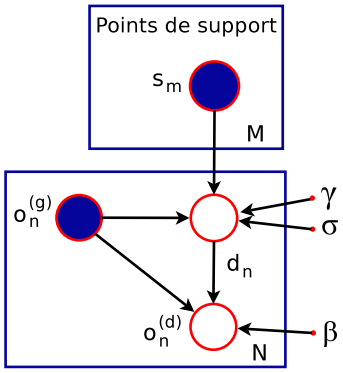


Figure 3.12 Modèle gaphique d'une approche probabiliste (voir Geiger *et al.*, 2010, p. 5)

La distribution préliminaire $p(d_n|S, o_n^{(g)})$ peut prendre la forme d'une combinaison d'une distribution uniforme et une distribution gaussienne échantillonnée. (voir Geiger *et al.*, 2010, p. 6)

$$p(o_n^{(d)}|S, o_n^{(g)}) \propto \begin{cases} \gamma + \exp(-\dfrac{(d_n - \mu(S, o_n^{(g)}))^2}{2\sigma^2}) & \text{si } |d_n - \mu| < 3\sigma \vee d_n \in N_S \\ 0 & \text{sinon} \end{cases} \tag{3.48}$$

$\mu(S, o_n^{(g)})$ est exprimée comme un group de fonctions linéaires qui interpolent les disparités en employant l'algorithme de triangulation Delaunay sur les points de support.

La vraisemblance d'image peut être écrite comme une distribution laplacienne contrainte : (voir Geiger *et al.*, 2010, p. 6)

$$p(o_n^{(d)}|o_n^{(g)}, d_n) \propto \begin{cases} \exp(-\beta \, ||f_n^{(g)} - f_n^{(d)}||) & \text{si } \begin{pmatrix} u_n^g \\ v_n^g \end{pmatrix} = \begin{pmatrix} u_n^d + d_n \\ v_n^d \end{pmatrix} \\ 0 & \text{sinon} \end{cases} \tag{3.49}$$

où $f_n^{(g)}$ et $f_n^{(d)}$ sont vecteurs de caractéristique de l'image de gauche et celle de droite respectivement. β est une constante. (voir Geiger *et al.*, 2010, p. 6)

La carte de disparité peut être calculé par : (voir Geiger *et al.*, 2010, p. 7)

$$d_n^* = \text{argmax } p(d_n|o_n^{(g)}, o_1^{(d)}, \ldots, o_N^{(d)}, S) \tag{3.50}$$

où $o_1^d, \ldots, o_N^d$ désignent toutes les observations de l'image de droite sur la ligne épipolaire de o_n^g. La distribution postérieure peut être factorisée comme : (voir Geiger *et al.*, 2010, p. 7)

$$p(d_n|o_n^{(g)}, o_1^{(d)}, \ldots, o_N^{(d)}, S) \propto p(d_n|S, o_n^{(g)})\, p(o_1^{(d)}, \ldots, o_N^d|o_n^{(g)}, d_n) \tag{3.51}$$

Tandis que : (voir Geiger *et al.*, 2010, p. 7)

$$p(o_1^{(d)}, \ldots, o_N^d|o_n^{(g)}, d_n) \propto \sum_{i=1}^{N} p(o_i^{(d)}|o_n^{(g)}, d_n) \tag{3.52}$$

Alors nous avons une fonction d'énergie : (voir Geiger *et al.*, 2010, p. 7)

$$E(d) = \beta\, ||f^{(g)} - f^{(d)}(d)|| - \log[\gamma + \exp(-\frac{[d - \mu(S, o^{(g)})]^2}{2\sigma^2})] \tag{3.53}$$

$f^{(d)}(d)$ est le vecteur de caractéristique au pixel $(u^{(g)} - d, v^{(g)})$. (voir Geiger *et al.*, 2010, p. 7) Principalement parce que cette méthode évite de faire la mise en correspondance par pixel et elle s'appuie sur la fiabilité des points de support, la qualité et la performance sont bonnes, meilleures que plusieurs méthodes. (voir Geiger *et al.*, 2010)

3.13.3 Modèle de Markov caché

Le modèle de Markov caché (MMC) joue un rôle important dans l'apprentissage par machine. La carte de disparité est un ensemble de distance des pixels correspondants dans deux images. Il arrive souvent qu'un point ne soit pas indépendant des autres points dans une image. La forme et le changement de forme selon le point de vue doivent être considérés. Par convention, nous supposons que chaque pixel dépend du pixel de gauche sur une ligne de disparité. Nous aurons un modèle de MMC pour chaque ligne horizontale de pixels illustrés dans la Figure 3.13

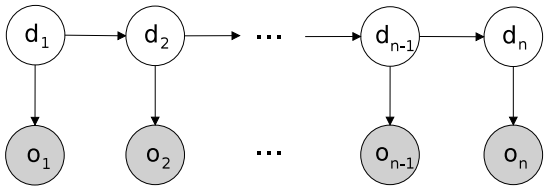


Figure 3.13 Un MMC horizontal concernant le problème de disparités des images

La variable d_i représente la disparité de pixels optimale dans une ligne de carte de disparités. La variable observée o_i est la distance minimale entre deux pixels de deux images

concernées. La probabilité jointe des variables observées et cachées est comme ci-dessous :

$$P(\{o\},\{d\}) = \prod_{t=1}^{\mathrm{T}} P(o_t|d_t) \prod_{t=1}^{\mathrm{T}} P(d_{t+1}|d_t)P(d_t)$$

Un MMC a cinq éléments essentiels : (voir Rabiner, 1989, pp. 260–261).

1. N, le nombre d'états dans le modèle. Un ensemble d'états individuels est $S = \{S_1, S_2, \ldots, S_N\}$, un état au moment t est d_t.
2. M, le nombre de symboles distincts par état. Un ensemble de symboles individuels est $O = \{o_1, o_2, \ldots, o_M\}$.
3. La distribution probabiliste d'états est définie comme $A = \{a_{ij}\}$ où $a_{ij} = P[d_{t+1} = S_j \mid d_t = S_i], 1 \leq i, j \leq N$.
4. La distribution probabiliste de symboles d'observation dans l'état j, $B = \{b_j(k)\}$, où $b_j(k) = P[o_k \text{ à } t \mid o_t = S_j], 1 \leq j \leq N, 1 \leq k \leq M$.
5. La distribution initiale des états $\pi = \{\pi_i\}$ où $\pi_i = P[d_1 = S_i], 1 \leq i \leq N$.

Le modèle est normalement noté $\lambda = (A, B, \pi)$ (voir Rabiner, 1989, pp. 260–261).

L'algorithme Viterbi (voir Viterbi, 1967, pp. 264–265) est employé dans le décodage de code convolutif utilisé dans la télécommunication. Il est aussi couramment utilisé dans la reconnaissance de parole. La complexité de calcul est passée de simple recherche exhaustive M^N à NM^2 (voir Snyder et Qi, 2004, p 25).

Déjà en 1975, Baker a présenté un système de compréhension de la parole nommé DRAGON en employant le MMC (voir Baker, 1975).

3.13.4 Modèle de généralisation de MMC

Le MMC présenté dans la sous-section 3.13.3 suppose que sur chaque ligne de balayage, un pixel de la carte de disparité influence la valeur du pixel voisin direct à droite et l'observation. En réalité, un pixel d'une image dépend des valeurs voisines autour. Une première généralisation de MMC est de passer de connexion de gauche à droite à celle grillée. La Figure 3.14 montre cette généralisation et une partie de passage de message en employant la méthode de graphe de facteur de Kschischang *et al.*(voir Kschischang *et al.*, 2001).

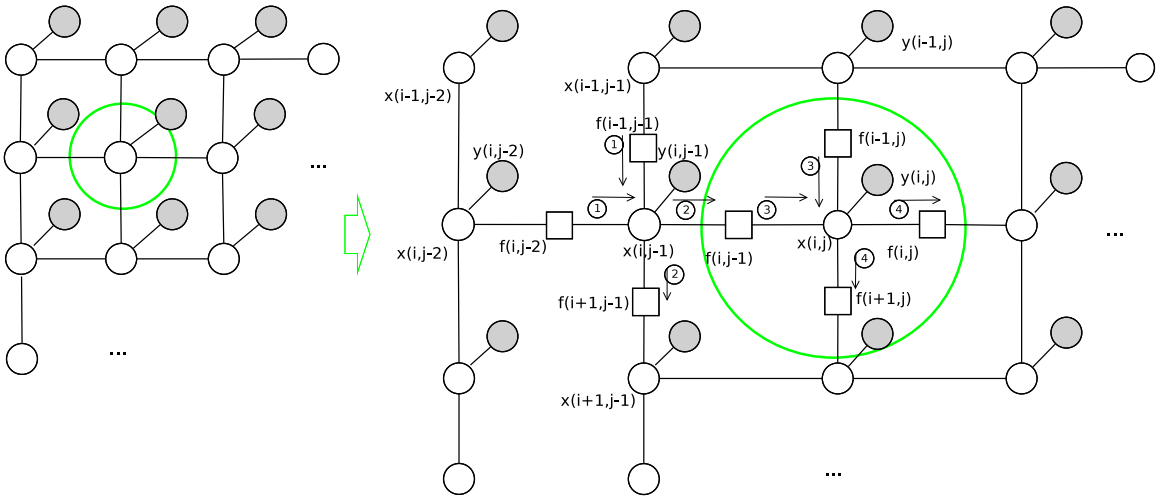


Figure 3.14 Généralisation de MMC

Supposons qu'une exécution de propagation de croyance passe de gauche à droite et de haut en bas et le point $x(i,j)$ est l'étape actuelle. La première étape de passage de message est :

$$\begin{aligned} m_{f_{(i-1,j-1)}\to x(i,j-1)} &= \sum_{\backsim\{x(i,j-1)\}} m_{x(i-1,j-1)\to f_{(i-1,j-1)}} f_{(i-1,j-1)}(x[i,j-1]) \\ &= m_{x(i-1,j-1)\to f_{(i-1,j-1)}} f_{(i-1,j-1)}(x[i,j-1]) \\ m_{f_{(i,j-2)}\to x(i,j-1)} &= \sum_{\backsim\{x(i,j-1)\}} m_{x(i,j-2)\to f_{(i,j-2)}} f_{(i,j-2)}(x[i,j-1]) \\ &= m_{x(i,j-2)\to f_{(i,j-2)}} f_{(i,j-2)}(x[i,j-1]) \end{aligned} \tag{3.54}$$

La deuxième étape est :

$$\begin{aligned} m_{x(i,j-1)\to f_{(i,j-1)}} &= m_{f_{(i-1,j-1)}\to x(i,j-1)}\ m_{f_{(i,j-2)}\to x(i,j-1)} \\ &\quad\ m_{f_{(i+1,j-1)}\to x(i,j-1)}\ m_{y(i,j-1)\to x(i,j-1)} \\ m_{x(i,j-1)\to f_{(i+1,j-1)}} &= m_{f_{(i-1,j-1)}\to x(i,j-1)}\ m_{f_{(i,j-2)}\to x(i,j-1)} \\ &\quad\ m_{f_{(i,j-1)}\to x(i,j-1)}\ m_{y(i,j-1)\to x(i,j-1)} \end{aligned} \tag{3.55}$$

La troisième étape est :

$$\begin{aligned} m_{f_{(i-1,j)}\to x(i,j)} &= \sum_{\backsim\{x(i,j)\}} m_{x(i-1,j)\to f_{(i-1,j)}} f_{(i-1,j)}(x[i,j]) \\ &= m_{x(i-1,j)\to f_{(i-1,j)}} f_{(i-1,j)}(x[i,j]) \\ m_{f_{(i,j-1)}\to x(i,j)} &= \sum_{\backsim\{x(i,j)\}} m_{x(i,j-1)\to f_{(i,j-1)}} f_{(i,j-1)}(x[i,j]) \\ &= m_{x(i,j-1)\to f_{(i,j-1)}} f_{(i,j-1)}(x[i,j]) \end{aligned} \tag{3.56}$$

La quatrième étape est :

$$\begin{aligned} m_{x(i,j)\to f_{(i,j)}} = \; & m_{f_{(i-1,j)}\to x(i,j)}\, m_{f_{(i,j-1)}\to x(i,j)} \\ & m_{f_{(i+1,j)}\to x(i,j)}\, m_{y(i,j)\to x(i,j)} \\ m_{x(i,j)\to f_{(i+1,j)}} = \; & m_{f_{(i-1,j)}\to x(i,j)}\, m_{f_{(i,j-1)}\to x(i,j)} \\ & m_{f_{(i,j)}\to x(i,j)}\, m_{y(i,j)\to x(i,j)} \end{aligned} \tag{3.57}$$

Après cette étape, le flux passe de droite à gauche et de bas en haut.

Cette généralisation de MMC est en fait un modèle de champ aléatoire de Markov (MRF). La méthode de passage de message est appelée la propagation de croyance. Nous pouvons espérer avoir un bon résultat avec ce modèle. Et la propagation de coûts ressemble à la méthode d'agrégation des coûts de SGM (*semi-global matching*). Toutefois, le temps de calcul sera élevé et l'augmentation de précision au détriment du temps de calcul n'est pas le travail principal de cette thèse. Nous allons essayer de rapprocher cette méthode avec une méthode de multiples passes de MMC qui a une implémentation rapide en GPGPU.

3.14 Traitement après le calcul initial

Gibson a observé l'importance de perception des relations entre les figures et le fond. (voir Gibson, 1950) D'où vient l'importance d'avoir une surface continue. Si une scène n'ayant que des points clés pour s'identifier, c'est principalement par l'apprentissage que nous avons pu lui donner le sens. Des significations sont innées et ne sont pas apprises. (voir Gibson, 1950, pp. 206–212)

3.14.1 Surface visible

L'importance de forme est évidente. C'est pourquoi qu'il y a des approches sophistiquées globales d'interpolation. Par exemple une approche informatique de représentation de surface visible de Terzopoulos, dont le processus de reconstruction de surface visible unifie quatre

buts : (voir Terzopoulos, 1988)

- Inégration : intégration des contraintes provenant de sources multiples.
- Interpolation : propagation des informations de formes intégrées vers les régions manquant d'information de formes.
- Discontinuités : détection des discontinuités de surface.
- Efficacité : l'emploi de la relaxation multirésolution parmi une hiérarchie de représentations de surface pour accélérer la reconstruction.

Si $\mathfrak{F}$ est l'espace linéaire de la fonction admissible, $\mathcal{S}(v)$ est une fonction de stabilisation qui mesure le lissage d'une fonction $v \in \mathfrak{F}$, et $\mathcal{P}(v)$ est une fonction de pénalité sur $\mathfrak{F}$ qui mesure la différence entre v et les contraintes données, on peut formuler le problème de reconstruction de surface visible régularisée selon le principe variationnel suivant : (voir Terzopoulos, 1988, p. 418)

$$\text{Trouver } u \in \mathfrak{F} \text{ pour que } \mathcal{E}(u) = \inf_{v \in \mathfrak{F}} \mathcal{E}(v), \text{où la fonction d'énergie } \mathcal{E}(v) = \mathcal{S}(v) + \mathcal{P}(v). \tag{3.58}$$

Une approche multiniveau est recommandée, voir son rapport pour plus de détails. (voir Terzopoulos, 1982)

3.14.2 Interpolation basée sur segments

Hirschmüller a proposé une interpolation basée sur l'information de segmentation. Une interpolation bilinéaire sera appliquée à l'intérieur des segments, tandis qu'une valeur minimale est employée pour interpoler en fonction des segments. Cette dernière méthode sera acceptable pour les petites zones invalides. Pour les autres types de zone, il faut faire un compromis entre la simplicité de méthode et le résultat souhaité. (voir Hirschmüller, 2003, pp. 34–35)

Pour chaque disparité invalide d_{ik}, on cherche les plus proches disparités en haut d_{ih}, en bas d_{ib}, à gauche d_{gk} et à droite d_{dk}. S_{ik} est le segment où se trouve la disparité d_{ik}. d_{ik} est déterminé par : (voir Hirschmüller, 2003, pp. 34–35)

$$d_{ik} = \begin{cases} \frac{d_h + d_v}{2} & \text{si } S_h = Sv, \\ \min(d_h, d_v) & \text{si } S_h \neq Sv \end{cases} \tag{3.59}$$

où

$$d_h = \begin{cases} \frac{(d_{dk} - d_{gk})(i-g)}{d-g} + d_{gk} & \text{si } S_{gk} = S_{dk}, \\ \min(d_{gk}, d_{dk}) & \text{si } S_{gk} \neq S_{dk} \end{cases} \tag{3.60}$$

$$d_v = \begin{cases} \frac{(d_{ib}-d_{ih})(k-h)}{b-h} + d_{ih} & \text{si } S_{ih} = S_{ib}, \\ \min(d_{ih}, d_{ib}) & \text{si } S_{ih} \neq S_{ib} \end{cases} \tag{3.61}$$

$$S_h = \begin{cases} S_{gk} & \text{si } d_{gk} < d_{dk}, \\ S_{dk} & \text{si } d_{gk} \geq d_{dk} \end{cases} \tag{3.62}$$

$$S_v = \begin{cases} S_{ih} & \text{si } d_{ih} < d_{ib}, \\ S_{ib} & \text{si } d_{ih} \geq d_{ib} \end{cases} \tag{3.63}$$

3.15 Méthodes pour obtenir une carte de disparité de référence

Mouaddib *et al.* ont présenté un article sur les méthodes de capture des images stéréoscopiques. (voir Mouaddib *et al.*, 1997). Batlle *et al.* ont une version plus détaillée. (voir Batlle *et al.*, 1998) Souvent, on emploie la lumière structurée pour résoudre le problème de correspondance. La méthode consiste de remplacer un deuxième appareil photo par une source de lumière qui projette de la lumière de structure connue puis étudie la correspondance entre la lumière capturée et celle projetée. (voir Batlle *et al.*, 1998, p. 965) Déjà en 1971, Shirai et Suwa ont introduit une méthode de projeter une lumière sur un objet étudié puis un appareil photo capture chaque mouvement. (voir Shirai et Suwa, 1971; Shirai, 1972)

Scharstein *et al.* ont présenté une méthode en employant de la lumière structurée qui consiste de lancer des motifs lumineux structurés sur la scène et de déterminer une étiquette pour chaque pixel par appareil photo. (voir Scharstein *et al.*, 2003)

ZALEVSKY *et al.* ont inventé le système de projection de motif aléatoire au laser et de correspondance de relief capturé et le motif projeté afin d'extraire d'information 3D. Ce système est la base du Kinect qui est un périphérique de Microsoft permettant de contrôler des jeux vidéo sans utiliser de manette. Le système peut produire une carte de profondeur presque en temps réel et peut s'adapter à plusieurs situations selon la configuration. La distance de capture peut être grande. L'emploi de motif aléatoire est pour contrer la limite de motif répétitif pour qui la distance maximale de correspondance doit être moins de la largeur de chaque élément du motif répétitif. (voir ZALEVSKY *et al.*, 2007)

Mohan *et al.* ont employé de lumière structurée de diode électroluminescente (LED) et une seule image pour obtenir les informations de profondeur sous condition que la gamme de profondeur est prédéfinie. (voir Mohan *et al.*, 2011)

3.16 Analyse

Nous avons recensé plusieurs méthodes qui ont des liens directs avec nos études ou qui les ont inspirées. Les méthodes revues sont des méthodes de calcul direct au lieu de méthodes globales qui sont normalement plus lentes. Notre but est d'avoir une approche rapide. Nous voulons employer une combinaison de Census et SDC pour profiter des avantages des deux approches de calcul de coût, nous pouvons aussi employer l'approche d'agrégation de coûts de SGM pour combiner des informations du voisinage afin d'améliorer la qualité.

Nous n'avons pas fait l'interpolation bilinéaire parce que dans la zone de bordure, il est difficile de préciser la dégradation de disparité, aussi avons-nous utilisé la méthode de remplissage des disparités des images en plus haute définition. Cette idée de remplissage des disparités d'une grande carte de disparité a un lien direct avec celle de méthodes hiérarchiques, sauf que notre méthode cherche à améliorer la petite carte avec les informations provenant d'une grande carte.

3.17 Conclusion

Parce que nous n'avons pas employé les méthodes de calcul global, il est difficile d'avoir une qualité élevée de correspondance. Des compromis sont toujours possibles et nous en avons exploité dans une certaine mesure. Nous allons combiner les idées de ces méthodes pour essayer d'améliorer la qualité de carte de disparité et la performance de sa création.

CHAPITRE 4

ÉTUDES DE CARTE DE DISPARITÉ

4.1 Environnement de tests et images stéréoscopiques employées

Nous avons employé un NVIDIA® GeForce® GTX 580 et un Intel® Core™ i7-2600K @ 3,40 GHz sous Windows 7 Professional. Intel® Core™ i7-2600K a 4 cœurs, mais l'exécution de référence en CPU s'exécute sur un seul cœur. Pour les tests en employant le GPGPU, la taille de bloc pour le lancement des fils d'exécution en parallèle est 128. Le nombre de lignes traitées verticalement dans un bloc est de 41.

Durant ces études, nous avons employé principalement les images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Les images de 2005 employées incluent les plus grandes et les plus petites de « *Art* », de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». Pour ce qui est des images de 2003, elles incluent les images les plus grandes et les plus petites de « *Cones* » et de « *Teddy* ». Le Tableau 4.1 montre une liste des tailles d'images employées. Sauf dans le cas d'employer les deux fois plus grandes images où r est 2, normalement r est le ratio entre la taille des grandes images et celle des petites images, soit $r \in [3, 4]$.

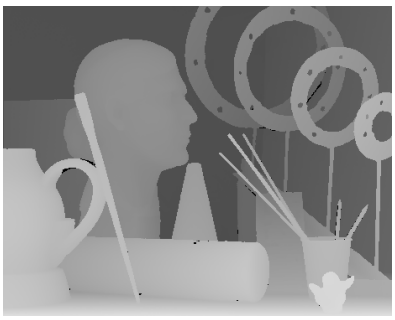
Tableau 4.1 Liste des images stéréoscopiques employées

Nom	Petites images	2 fois plus grandes images	Grandes images	r
Art	463×370	–	1390×1110	3
Books	463×370	–	1390×1110	3
Dolls	463×370	–	1390×1110	3
Laundry	447×370	–	1342×1110	3
Moebius	463×370	–	1390×1110	3
Reindeer	447×370	–	1342×1110	3
Cones	450×375	900×750	1800×1500	4
Teddy	450×375	900×750	1800×1500	4

La Figure 4.1 la Figure 4.2 affichent ces images stéréoscopiques de gauche et de droite et les cartes de disparité de la direction gauche à droite. (voir Blasiak *et al.*, 2005; Scharstein *et al.*, 2003)

image de gauche d'*Art*

image de droite d'*Art*



carte de disparité
de référence

image de gauche de *Books*

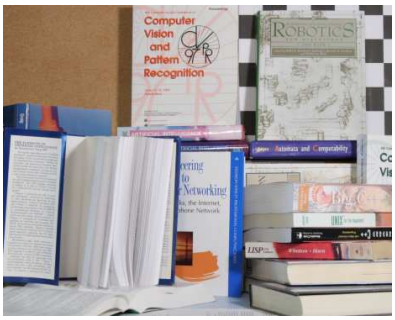
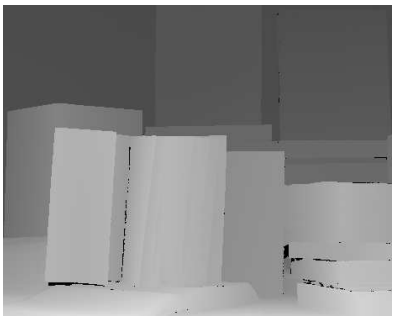


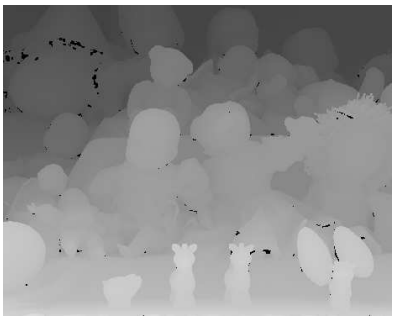
image de droite de *Books*



carte de disparité
de référence

image de gauche de *Dolls*

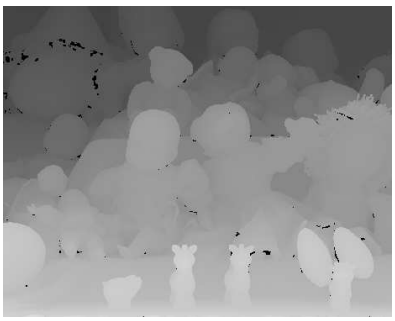
image de droite de *Dolls*



carte de disparité
de référence

image de gauche de *Laundry*

image de droite de *Laundry*

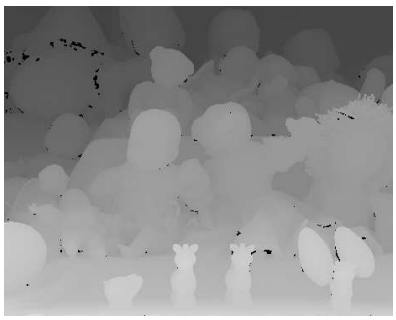


carte de disparité
de référence

Figure 4.1 Cartes de disparité (voir Blasiak *et al.*, 2005)

image de gauche de *Moebius*

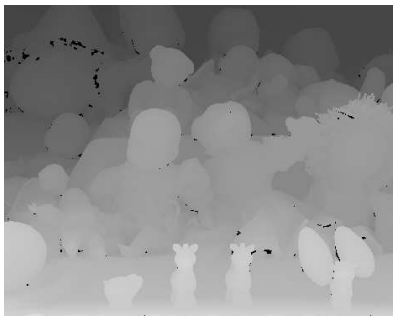
image de droite de *Moebius*



carte de disparité
de référence

image de gauche de *Reindeer*

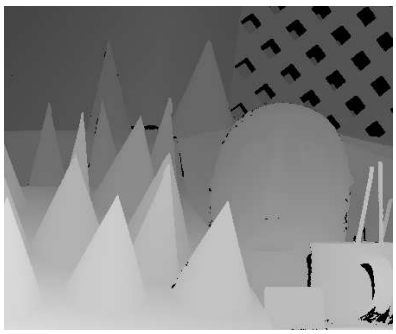
image de droite de *Reindeer*



carte de disparité
de référence

image de gauche de *Cones*

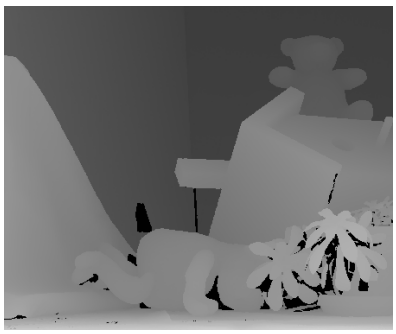
image de droite de *Cones*



carte de disparité
de référence

image de gauche de *Teddy*

image de droite de *Teddy*



carte de disparité
de référence

Figure 4.2 Cartes de disparité (voir Blasiak *et al.*, 2005; Scharstein *et al.*, 2003)

Les cartes de disparité de référence sont créées avec la technique de lumière structurée. (voir Blasiak *et al.*, 2005; Scharstein *et al.*, 2003) Ces cartes de référence sont complètes et denses, elles contiennent des disparités pour les zones qui ne sont visibles que pour une vue. Nous études emploient principalement la lumière passive, il y aura des zones que nous ne pouvons pas faire une correspondance de qualité. Donc principalement nous faisons l'estimation de taux d'erreur sur les pixels visibles. Nous avons calculé la carte des pixels visibles par la vérification de consistance en employant deux cartes de disparité de deux directions : gauche-droite / droite-gauche. Une carte des pixels visibles indique les pixels visibles pour deux vues, par conséquent elle indique aussi les pixels invisibles pour une vue. C'est pour cette raison que cette carte est appelée aussi la carte d'éclipse. La vérification de consistance détermine si les disparités calculées de deux directions sont cohérentes. Si la différence excède un seuil $s = 1$, nous traitons le pixel correspondant comme un pixel invisible. Nous avons prendre le seuil $s = 1$ parce que les cartes de disparité de référence sont dans la gamme [0, 255], calculées pour les grandes images puis réduites en taille. Or, nous calculons les disparités pour les petites images qui sont r ($r \in [3, 4]$) fois plus petites que les grandes images et nous calculons les cartes de disparité en nombre entier, nous avons besoin le seuil $s = 1$ pour contrer la perte de précision dans la conversion de données.

Suppose $I_g(x, y)$ est un pixel dans l'image de gauche. On fait la vérification de consistance en prenant une valeur de disparité de la carte de disparité de gauche à droite $d_{gd}(x, y)$, puis on calcule $x - d_{gd}(x, y)$, si $x - d_{gd}(x, y) \geq 0$, c'est la position du pixel correspondant dans l'image de droite et aussi dans la carte de disparité de droite à gauche $d_{dg}(x - d_{gd}(x, y), y)$ selon la définition de disparité. Si la différence absolue entre $d_{gd}(x, y)$ et $d_{dg}(x - d_{gd}(x, y), y)$ est moins du seuil s, le pixel $I_g(x, y)$ est cohérent dans deux cartes de disparité, sinon le pixel $I_g(x, y)$ est visible pour un seul appareil photo.

Les cartes des pixels visibles employées sont montrées dans la Figure 4.3. Les cartes de *Cones* et de *Teddy* sont fournies par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Les autres cartes sont créées avec les cartes de disparité de référence en mettant le seuil $s = 1$ (voir Blasiak *et al.*, 2005).

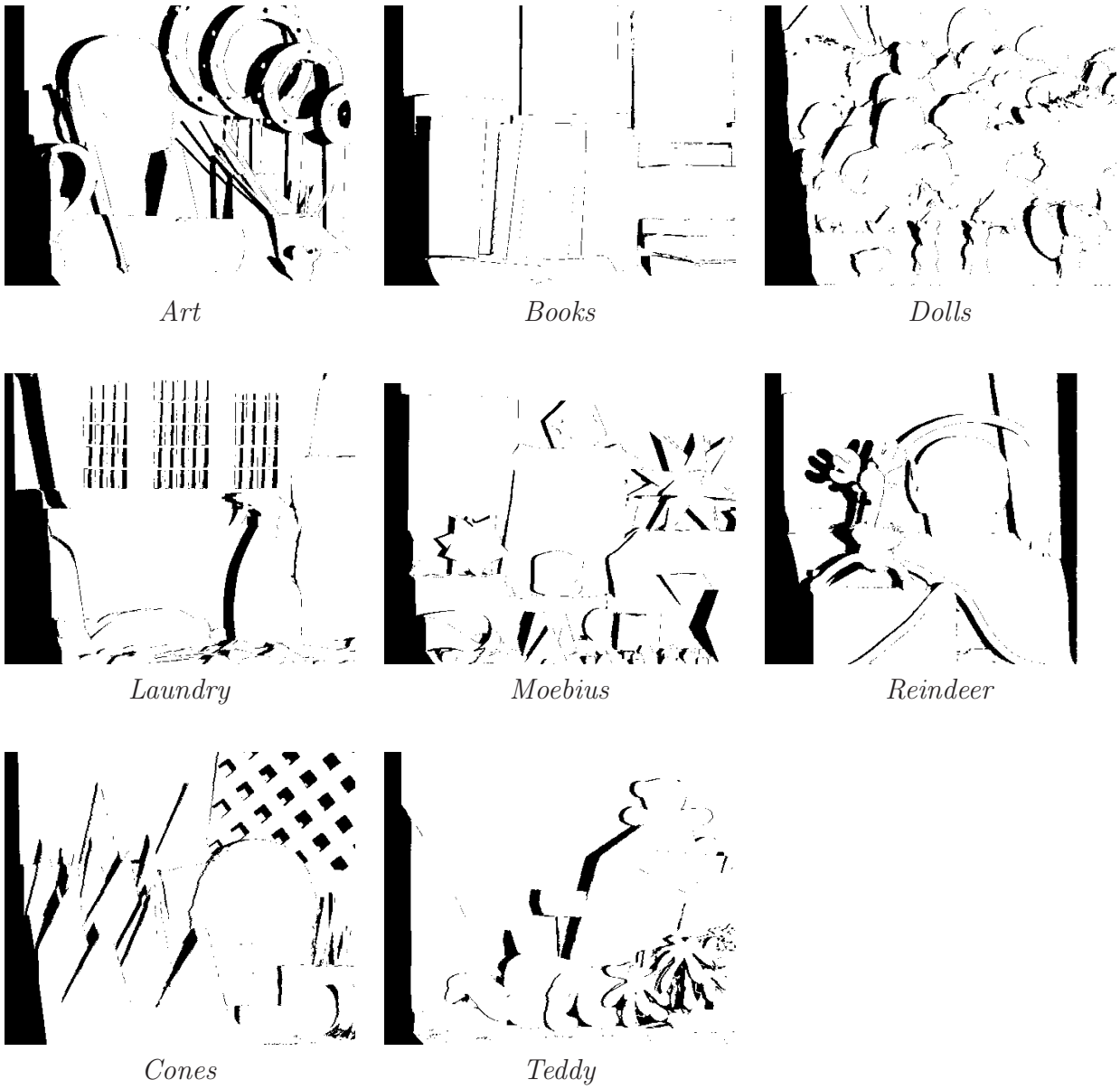


Figure 4.3 Cartes d'éclipse employées (voir Blasiak *et al.*, 2005; Scharstein *et al.*, 2003)

La plupart de cartes de disparité obtenues dans ce mémoire sont appliquées avec ces cartes d'éclipse pour n'afficher que les disparités des pixels visibles. Le pixel noir ayant la valeur 0 est souvent le résultat d'application de la carte d'éclipse.

4.2 Mise en correspondance par bloc

Nous voulons explorer le potentiel des images en haute définition dans ce mémoire et nous voulons aussi traiter des vidéos, nous avons une demande de traitement de temps réel

ou presque, à environs 60 fps (images par seconde). Nous avons choisi d'employer un GPGPU pour accélérer le traitement parce que c'est un moyen moins cher de traitement parallèle. Un traitement parallèle avec OpenMP ou Open MPI demande normalement plusieurs cœurs de traitement, coûte plus cher et a moins de possibilités d'être réalisé dans un système embarqué.

Nous définissons une fenêtre de comparaison qui est normalement un carré ou un rectangle horizontal proche d'un carré de sorte que le pixel concerné se trouve au centre du carré ou du rectangle. Nous avons besoin de deux paramètres rH, rV pour définir le nombre de pixels à ajouter horizontalement et verticalement autour de chaque pixel. La différence de valeur du paramètre horizontal rH et du paramètre vertical rV doit être au maximum 1 pixel pour garder la forme plus proche d'un carré. Le but ici est de regrouper d'information autour d'un pixel, donc nous avons demandé la fenêtre d'être la plus proche possible à un carré afin d'éviter le plus possible de bruit. Évidemment, un cycle sera meilleur, mais un carré sera plus facile à traiter. La taille de fenêtre est calculée ainsi : $_f = (rH + rV + 1)$. Dans le but de simplifier nos études, nous n'employons que des carrés. Donc, nous n'avons qu'un paramètre r. Il en va de même dans l'équation 3.26.

Ensuite, nous appliquons cette fenêtre à l'image de gauche (imgG) et à l'image de droite (imgD). Prenons l'image de gauche comme image de référence. Supposons que le pixel en question est imgG(x, y), nous déplaçons la fenêtre à partir de imgD(x, y), chaque fois d'un pixel à droite jusqu'à une valeur maximale de déplacement d ou jusqu'à la bordure de l'image imgD, selon ce qui vient en premier. Soit une séquence de pixels : imgD(x, y), imgD$(x + 1, y)$ ... La valeur maximale de déplacement d est une valeur prédéfinie qui équivaut à la disparité maximale prise en compte. Cette valeur maximale permet d'abandonner des valeurs hors du champ considéré.

À chaque déplacement, nous calculons la somme des différences quadratiques des pixels correspondants dans cette fenêtre de comparaison, soit

$$\sum_{0 \leq i \leq d} \{\text{imgG}(x, y) - \text{imgD}(x + i, y)\}^2.$$

La valeur de déplacement qui correspond à la somme minimale sera la disparité du pixel (x, y).

Nous avons implémenté l'algorithme présenté dans le rapport de Stam en Open CL. Nous n'avons pas employé la texture comme celle en CUDA C. Dans nos études, la taille de fenêtre f est T_f, c'est-à-dire que la largeur et la hauteur de f sont toutes T_f dans nos études. Le nombre horizontal de pixels de chaque côté d'un pixel est $l_f = \text{plancher}(T_f/2)$; le nombre vertical de pixels sera $h_f = T_f - l_f - 1$. La fonction plancher(y) donne un nombre entier qui est le plus élevé parmi les nombres inférieurs à y. Dans nos études, nous exigeons que T_f soit impair

pour simplifier la question, alors la fenêtre sera un carré défini par $h_f = l_f = \text{plancher}(T_f/2)$.

4.3 Trois méthodes d'implémentation de la mise en correspondance par bloc en GPGPU

Nous avons réalisé trois méthodes en GPGPU dans les essais d'accélération. La Méthode 0 emploie la mémoire globale pour accumuler les coûts. La Méthode 1 est une réalisation en OpenCL d'une implémentation de la mise en correspondance par bloc en GPGPU d'OpenCV mentionnée dans la sous-section 3.8.2, qui emploie la mémoire partagée pour calculer la valeur minimale et l'indice correspondant parmi un nombre prédéfini de disparités potentielles. Pour faciliter le traitement, cette Méthode ignore les zones de bordure. La Méthode 2 est basée sur la Méthode 1, mais qui traite aussi les zones de bordure au lieu de les ignorer comme le cas de la Méthode 1 afin d'augmenter la précision pour le cas de la vérification de consistance et de rendre la carte de disparité plus complète. La vérification de consistance s'opère en employant une carte de disparité calculée de gauche à droite et une autre carte calculée de droite à gauche.

La Méthode 0 est la base de notre réalisation. Nous calculons la somme e des écarts quadratiques de chaque colonne d'une fenêtre de comparaison f et nous les sauvegardons dans la mémoire globale. Ensuite nous calculons une somme s des éléments e d'un nombre de colonnes de la fenêtre de comparaison f. Cette somme s sera la somme des écarts quadratiques selon la fenêtre de comparaison des deux images en question.

Cette séparation de calcul en deux étapes peut contribuer à diminuer le nombre de calculs des écarts quadratiques parce qu'une colonne des écarts a été calculée puis partagée entre les fenêtres concernées. Initialement, le nombre de calculs est : T_f^2 multiplications, plus $(T_f - 1) \times (T_f + 1) = 2 \times T_f^2 - 1$ additions (ou soustractions). Une fois réalisée la séparation en deux étapes, le nombre de calculs pour chaque fenêtre sera : T_f multiplications, plus $T_f + 2 \times (T_f - 1) = 3 \times T_f - 2$ additions (ou soustractions). Comme une multiplication prend plus de temps qu'une addition, l'accélération est évidente. Voir la Figure 3.9 pour cette méthode dans la section 3.9.

Les produits d'écart quadratique se recouvrent horizontalement et verticalement parce que les fenêtres adjacentes se recouvrent elles-mêmes de la même façon. Écart quadratique est la différence quadratique, qui prend souvent la forme : $(\text{Img}_{ij} - \text{Img}_{kj})^2$. Pour illustrer cette caractéristique, supposons que nous sommes dans la ligne l, et que nous avons déjà calculé les sommes d'écart quadratique de la ligne précédente $l - 1$. Alors, nous calculons les sommes des colonnes des lignes suivantes en soustrayant les premiers éléments $l - h_f - 1$ des sommes de la ligne précédente, et en ajoutant les nouveaux éléments $l + h_f$ de la ligne l. Si

nous avons T_f multiplications plus $2 \times T_f - 1$ additions (ou soustractions) pour le calcul d'une colonne, l'ajout et la soustraction seront 2 multiplications plus 4 additions (ou soustractions). Nous avons illustré cette méthode dans la Figure 3.10

Le problème de cette méthode est la lenteur de l'accès à la mémoire globale. Même s'il n'y a pas eu de conflit d'accès pour une même adresse par différents processus, l'accès à la mémoire globale lui-même et la synchronisation obligatoire avant l'addition des sommes de colonne entraîne une détérioration de performance.

En nous inspirant d'une implémentation de la mise en correspondance par bloc en GPGPU d'OpenCV mentionnée dans la sous-section 3.8.2, nous avons réalisé l'implémentation « Méthode 1 ». Cette méthode utilise le principe d'accélération par partage de résultats intermédiaires de la Méthode 0. De plus, elle calcule un indice qui correspond à la valeur minimale du coût de correspondance parmi un nombre n_d des disparités potentielles pour chaque pixel à l'aide de la mémoire partagée. Au cours d'une étape de calcul, un nombre n_d de disparités potentielles sera calculé puis le coût minimal et la valeur d'indice correspondante seront retenus pour continuer la comparaison. Cela signifie que la flexibilité sera compromise parce que la disparité maximale doit être une multiplication du nombre n_d. Afin de simplifier, les pixels n'ayant pas assez de fiabilité de disparité seront invalidés. Si maxDisparité désigne la disparité maximale, l_i et h_i désignent la largeur et la hauteur des pairs d'images en question ; pour un calcul de disparité de gauche à droite, on ne calcule que la zone en largeur et en hauteur définie par $[\text{maxDisparité} + l_f, l_i - l_f - 1]$ et $[h_f, h_i - h_f - 1]$. Les crochets "[" et "]" signifient que nous pouvons prendre le début et la fin de la portée, les chiffres sont présentés selon la convention du langage C, soit le chiffre commence par 0 et se termine par la valeur maximale moins 1.

En contrepartie, la disparité de droite à gauche ne calculera que la zone en largeur et en hauteur définie par $[w_f, l_i - \text{maxDisparité} - l_f - 1]$ et $[h_f, h_i - h_f - 1]$. Bien sûr, si nous employons les cartes de disparité en pleine taille pour faire la vérification de consistance basée sur la disparité de gauche à droite et de droite à gauche, cette méthode va produire plus de valeurs invalides dans la zone $[l_i - \text{maxDisparité} - l_f, l_i - l_f - 1]$. Cela peut nuire à notre but initial de créer une carte de disparité assez proche de la vraie carte.

Durant nos études, nous calculons les taux d'erreur dans la zone définie $[w_f, l_i - \text{maxDisparité} - l_f - 1]$ et $[h_f, h_i - h_f - 1]$ comme ci-dessus pour ne pas être influencés par les zéros mis intentionnellement comme valeurs invalides. Donc, il faut garder en mémoire cette stratégie pour ne pas avoir une impression erronée en voyant les chiffres.

Puisque la Méthode 0 produit des disparités avec une certaine exactitude en zone considérée sans fiabilité, nous avons combiné les Méthodes 0 et 1 pour produire une Méthode 2 qui calcule un indice comme valeur intermédiaire parmi un nombre n_d des disparités potentielles

pour chaque pixel à l'aide de mémoire partagée. Le coût minimal et la valeur de disparité correspondante seront retenus pour continuer la comparaison. Ce point est identique à celui de la Méthode 1 sauf qu'avec la Méthode 2 nous calculons non seulement la zone qui offre le plus de fiabilité, mais aussi d'autres zones pour créer une carte de disparité complète. Bien sûr, les disparités dans ces dernières zones n'ont pas assez de fiabilité. La vérification de consistance nous aidera à les préciser.

4.4 Implémentation de la mise en correspondance par bloc en CPU comme référence dans les comparaisons de performance

L'implémentation de programme de référence pour la comparaison de performance est en CPU. L'algorithme est tout à fait proche de celui qu'on emploie dans GPGPU :

1. Rembourrons les images de gauche ($\text{img}_{\text{gauche}}$) et de droite ($\text{img}_{\text{droite}}$) pour faciliter le traitement aux bordures d'image.
2. Pour chaque possible valeur de disparité d :

   a. Déplaçons à droite l'image droite rembourrée le nombre de pas selon d pour obtenir une image rembourrée déplacée $\text{img}_{\text{déplacée}}$.

   b. Calculons une matrice m_{diff} avec chaque élément comme le produit de la différence entre les pixels de l'$\text{img}_{\text{gauche}}$ et ceux de l'$\text{img}_{\text{déplacée}}$. L'élément de la matrice m_{diff} peut être la somme des différences au carré (SDC) ou la somme des différences absolues (SDA).

   c. Accumulons des m_{diff} sur une fenêtre pour avoir une matrice $\text{m}_{\text{accumulé}}$. Nous calculons le premier élément comme une somme des éléments de m_{diff} sur une fenêtre carrée.

   d. Ensuite, nous soustrayons la première colonne de m_{diff} de gauche à l'intérieur du bloc et ajoutons une colonne de m_{diff} à droite du bloc sur un élément de la matrice $\text{m}_{\text{accumulé}}$ pour calculer un élément suivant de la matrice $\text{m}_{\text{accumulé}}$ en direction horizontale.

   e. Pour la ligne suivante, nous soustrayons la première ligne de m_{diff} de haut à l'intérieur du bloc et ajoutons une ligne de m_{diff} au-dessous du bloc pour calculer un élément suivant de la matrice $\text{m}_{\text{accumulé}}$ en direction verticale.

   f. Mettons à jour la valeur minimale de $\text{m}_{\text{accumulé}}$ et notons son indice.
3. Récupérons les indices finaux, qui sont les valeurs de disparité.

L'implémentation en CPU avec un seul cœur est prise comme référence où le calcul des écarts quadratiques est réduit au minimum afin de mieux comparer l'effet d'accélération.

4.5 Méthode d'évaluation dans nos études

Il faut noter que la carte de disparité de référence employée a une gamme de valeur de [0, 255]. Nos tests se font en nombre entier dans environ un tiers de la gamme totale. Dans la plupart de cas, nous avons fait les tests dans la gamme [0, 79] pour les images ayant la taille 1/3 fois de celle des plus grandes images, et la gamme [0, 63] pour les images ayant la taille 1/4 fois de celle des plus grandes images. Parce que nous avons besoin d'une gamme qui a un nombre pouvant être divisé par 8 compte tenu de particularités de nos Méthodes 1 et 2 et de notre choix de chiffre $n_d = 8$ comme nombre de possibilités dans un calcul expliqués dans la sous-section 4.3.

Ensuite, nous avons réduit les disparités de la carte de référence $C_{\text{référence}}$ à l'échelle de $1/r$ pour créer une carte de disparité $C_{\text{référenceRéduite}}$ afin de ramener les valeurs à la gamme [0, 79] ou [0, 63] que la carte obtenue C_{petite} se situe.

Puisqu'une différence de 1 peut être souvent négligée, nous avons calculé le ratio de pixels ayant la différence absolue plus que 1 dans la zone visible comme le taux d'erreur. Soit le ratio des pixels visibles pour des disparités dont la différence absolue par rapport à ceux de la carte de référence est supérieure à 1. La zone visible est désignée à l'aide d'une carte d'occlusion, qui est obtenue à partir de la vérification de consistance sur les cartes de disparité de référence de deux directions gauche-droite / droite-gauche en mettant le seuil de vérification de consistance à 0.

Si C est une carte de disparité et Réf est la carte de référence alors le taux d'erreur est l'équation suivante :

$$\frac{|\{\mathrm{C}_{ij} : \mathrm{C}_{ij} \text{ est visible } \& \ |\mathrm{C}_{ij} - \mathrm{Réf}_{ij}| > 1\}|}{|\{\mathrm{C}_{ij} : \mathrm{C}_{ij} \text{ est visible } \}|}. \tag{4.1}$$

Nous avons calculé le taux d'erreur dans la zone visible pour éviter l'interférence de pixels invisibles qui sont visibles pour au maximum un appareil photo parmi deux appareils photo. Par exemple, Scharstein et Szeliski a employé cette méthode d'évaluation. (voir Scharstein et Szeliski, 2002, p. 11) Si $d_C(x, y)$ est la carte de disparité calculée, $d_R(x, y)$ est la carte de disparité de référence et δ_d est un seuil de tolérance d'erreur de disparité, N est le nombre de pixels visibles, la définition de taux d'erreur est l'équation 4.2 (voir Scharstein et Szeliski, 2002, p. 11) :

$$B = \frac{1}{N} \sum_{(x,y)} (|d_C(x, y) - d_R(x, y)| > \delta_d) \tag{4.2}$$

Nous avons employé $\delta_d = 1$ comme plusieurs d'autres études (voir Scharstein et Szeliski, 2002; Szeliski et Zabih, 1999; Zitnick et Kanade, 2000).

Nous pouvons employer le comparatif d'égalité forte : deux pixels de deux images ayant exactement la même valeur sont considérés égaux, toutefois, deux cartes de disparité seront souvent considérées égales si la variation est assez limitée. Par exemple, Hirschmüller et Scharstein ont évalué les cartes de disparité au moyen d'un taux d'erreur qui compte le ratio du nombre de pixels visibles pour des disparités dont la différence absolue par rapport à ceux de la carte de référence est supérieure à 1 (voir Hirschmüller et Scharstein, 2009, p. 5).

Nous présentons la comparaison des méthodes en employant le taux d'erreur dans ce mémoire.

4.6 Temps d'exécution de la Méthode 0–2 avec la vérification de consistance

Nous avons montré le temps d'exécution de la Méthode 0–2 avec la vérification de consistance dans le Tableau 4.2.

Tableau 4.2 Temps d'exécution avec la vérification de consistance

Disparité maximale	Méthode	Taille de fenêtre	Temps d'exécution
les petites images			
79	Méthode 0	3	12,7 *ms*
		5	13,4 *ms*
	Méthode 1	3	4,2 *ms*
		5	4,2 *ms*
	Méthode 2	3	6,9 *ms*
		5	7,0 *ms*
les grandes images			
239	Méthode 0	9	189,7 *ms*
		15	221,4 *ms*
	Méthode 1	9	65,2 *ms*
		15	88,2 *ms*
	Méthode 2	9	119,0 *ms*
		15	149,1 *ms*

Nous pouvons faire le remplissage avec une stratégie sélective pour ne traiter que les pixels correspondants de ceux invalides dans une petite carte de disparité, cela peut éviter normalement 50% de calculs. Une étude plus profonde peut éventuellement encore réduire le temps d'exécution pour rendre le traitement de vidéos possible avec le principe de chercher plus d'information à partir des images en plus haute définition.

Nous avons comparé le taux d'accélération des trois Méthodes par rapport à la méthode accélérée en CPU en faisant une seule passe de gauche à droite de calcul de disparité dans la Figure 4.4. La valeur maximale de disparité est 79.

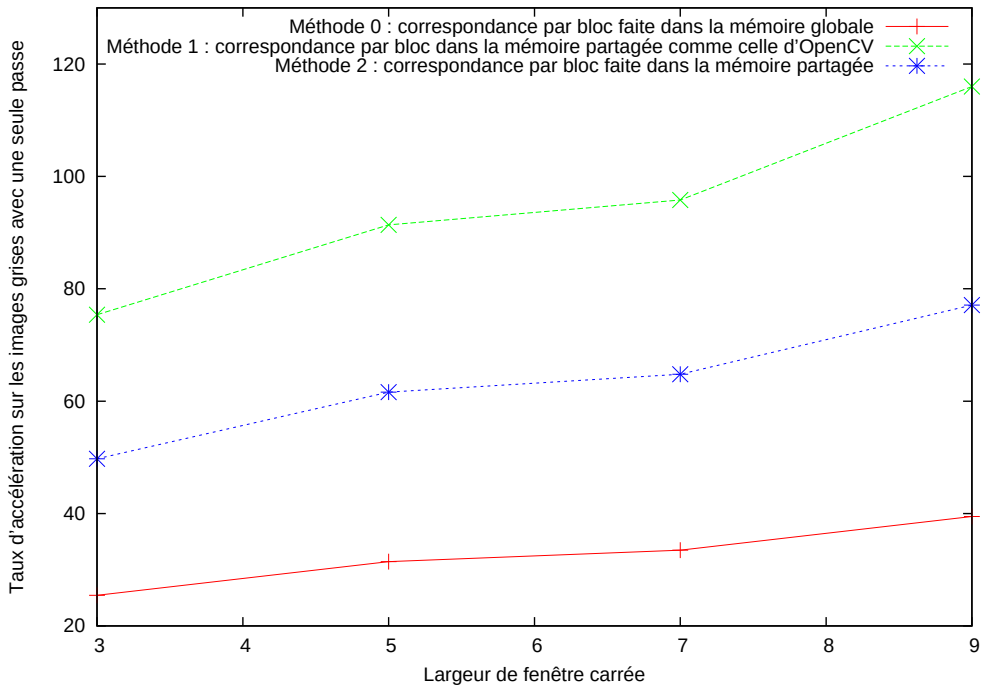


Figure 4.4 Taux d'accélération avec une seule passe selon trois Méthodes pour les images de 463×370

Le GPGPU a plusieurs processeurs qui contribuent en même temps au résultat final, ce qui explique l'accélération. Si la taille de la fenêtre est petite, l'effort de contrôle d'exécution et de synchronisation des fils d'exécution ralentit l'accélération en comparaison avec l'exécution sur CPU. Le GPGPU est d'une architecture SIMT, il va mieux de traiter des données en parallèle au lieu d'avoir plus d'essais de condition pour déterminer quelle branche à prendre. Dans ce dernier cas, un bloc de fil d'exécution va attendre un certain fil d'exécution même si les autres fils ne prennent pas d'action parce que ces exécutions ne satisfont pas la condition d'essai. Le principe pour mieux employer un GPGPU est de laisser les noyaux travailler autant que possible au lieu de les laisser attendre.

En moyenne, la Méthode 1 est 2 fois plus rapide que la Méthode 2, qui est elle-même 2 fois plus rapide que la Méthode 0 avec la taille d'images que nous avons employée. Bien que la Méthode 2 soit plus rapide que la Méthode 0 et les deux produisent le même résultat, la Méthode 2 exige que la disparité maximale soit un multiple de n_d pour profiter de l'accélération potentielle avec la mémoire partagée, nous devons garder la Méthode 0 pour plus de souplesse quant à la disparité maximale.

Nous avons également comparé les taux d'erreur des pixels visibles après une seule passe de calcul de disparité de gauche à droite dans la Figure 4.5. Les images employées sont les petites images de « *Art* » de Blasiak *et al.* (voir Blasiak *et al.*, 2005).

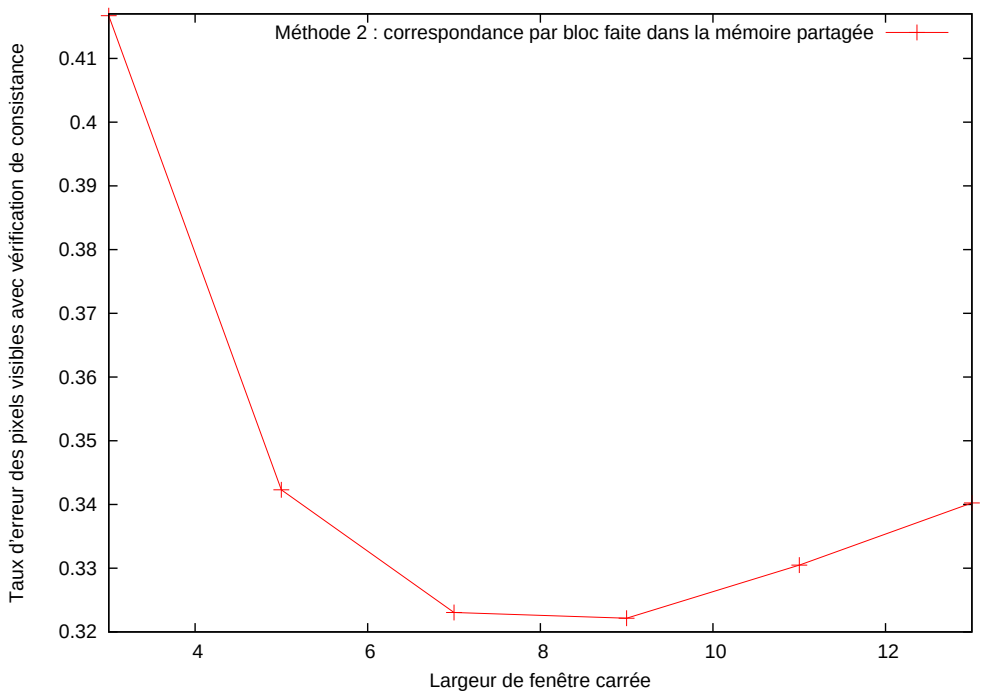


Figure 4.5 Taux d'erreur des pixels visibles avec la Méthode 2 sur les images de « *Art* »

La Méthode 2 est normalement plus rapide. Bien qu'elle présente des contraintes venant de la mémoire partagée et de la disparité maximale possible, normalement un peu de différence de disparité maximale ne change presque pas la précision. Nous avons employé cette Méthode dans la plupart de nos études. La Méthode 2 doit prendre un multiple de $n_d = 8$ (tel que défini dans nos études) comme fourchette de la gamme maximale de disparité, toutefois, parce que la Méthode 2 emploie la mémoire partagée pour travailler, si la taille de la fenêtre augmente trop, cette Méthode ne sera plus pratique. Dans ce cas, il faut soit limiter la valeur n_d, soit réduire la taille de bloc des fils d'exécution pour tenir compte de ces limites. Un dispositif de capacité de calcul 2.x de NVIDIA a 48 Ko de mémoire partagée. La Méthode 0 est plus robuste à cet égard parce qu'elle emploie la mémoire globale pour le calcul.

Nous pouvons observer qu'une augmentation de la taille de fenêtre de la comparaison peut augmenter la précision jusqu'à un certain point. Une fois dépassée une certaine taille de fenêtre, l'augmentation de la taille de la fenêtre va détériorer la précision. De la Figure 4.5, nous pouvons constater facilement que le taux d'erreur des pixels visible diminue avec l'augmentation de la taille de la fenêtre jusqu'à 7, ensuite le taux d'erreur va augmenter. Ce point de changement de tendance peut être considéré comme la meilleure taille de fenêtre pour ce type et cette taille d'images. Par exemple, 7 semble être la taille maximale pour le cas des images de « *Art* » de 463×370. La taille idéale pourrait être 5 si on considère tous les

aspects.

En fait, une trop grande fenêtre détériore la bordure d'objet à cause de la limite de la méthode de la mise en correspondance par bloc. Nous avons abordé cette question dans la sous-section 3.5.2, en indiquant que la corrélation stéréoscopique brouille la forme des objets avec une tendance sous-jacente à étendre horizontalement des objets (voir Hirschmüller, 2003, pp. 24, 26).

4.7 Méthode naïve de mise en correspondance par bloc

Dans la méthode de mise en correspondance par bloc améliorée, il y a beaucoup de synchronisation de fils d'exécution. Pour diminuer ces synchronisations, nous avons pensé faire le travail par une méthode naïve afin de mieux évaluer l'impact puis éventuellement améliorer la performance selon ces études.

Nous avons utilisé la méthode naïve de ligne en ligne d'images ou de pixel en pixel pour tester la méthode de mise en correspondance. Nous calculons la correspondance directement par les noyaux de GPGPU pour chaque pixel d'une ligne ou chaque pixel sans préciser l'ordre. Nous avons constaté que le taux d'accélération diminue rapidement avec l'augmentation de la taille de la fenêtre de comparaison.

La méthode naïve de mise en correspondance selon ligne balaie chaque ligne d'image, essaie de trouver le plus bas coût de correspondance. La comparaison se fait à l'aide d'une matrice dans la mémoire globale. Chaque fois que nous avons trouvé un plus bas coût, nous mettons à jour le coût global et l'indice correspondant. La méthode pour trouver l'indice avec le plus bas coût à l'aide d'une matrice globale est assez proche de celle employée dans les trois Méthodes précédentes.

La méthode naïve de pixel en pixel essaie de trouver directement la disparité de chaque pixel. Le calcul se fait selon pixel, donne directement le coût minimal et l'indice associé. Parce que le travail se fait dans la mémoire locale, à l'exception de lecture des images à comparer, nous pouvons constater que le taux d'accélération est supérieur à celui constaté avec la méthode naïve selon ligne.

La Figure 4.6 présente le résultat de comparaison de performance avec la Méthode 2 donnée dans la sous-section 4.3. La taille des images traitées est 463×370.

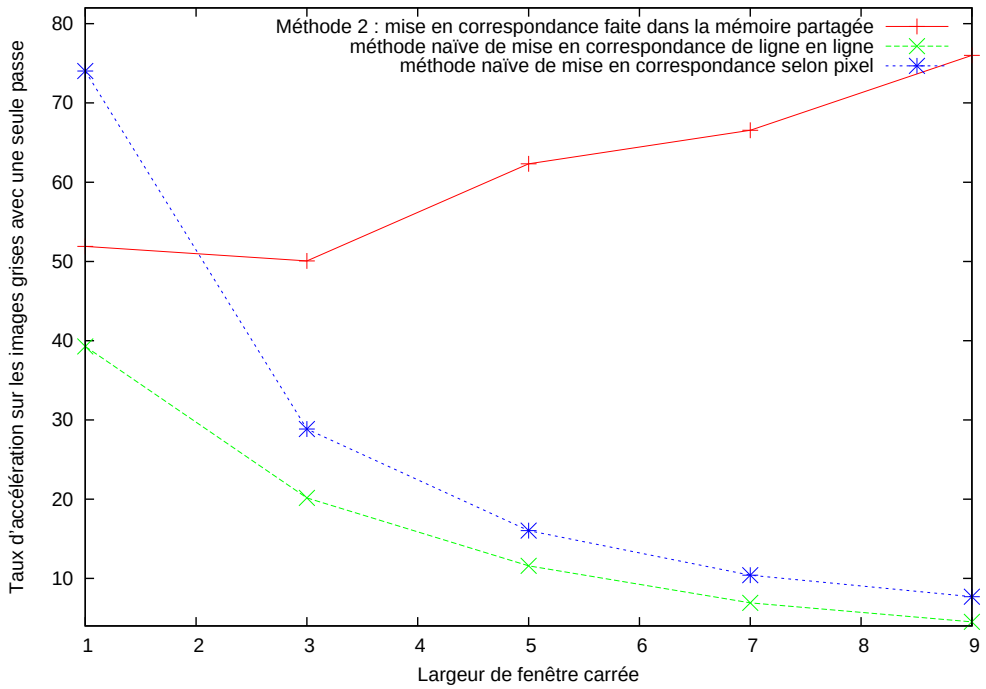


Figure 4.6 Taux d'accélération avec une seule exécution gauche-droite selon trois méthodes pour les images de 463×370

Nous avons constaté que le taux d'accélération pour de petites fenêtres est bon, mais pas pour de plus grandes fenêtres. Ces deux méthodes naïves font plusieurs fois la lecture avec la mémoire globale et calculent plusieurs fois l'écart quadratique. Ceci entraîne une mauvaise performance de calcul global. Néanmoins, le fait qu'il y ait une accélération, même si cela concerne plutôt les petites fenêtres, nous autorise à constater que le GPGPU accélère toujours le traitement. En effet, même si chaque noyau calcule le coût individuellement, il y a une accélération considérable notamment pour une petite fenêtre de comparaison. Il faut noter que le GPGPU employé ici est un produit haut de gamme pour que les exécutions avec de plus grandes fenêtres soient possibles.

Dans un cas extrême, par exemple, pour le cas d'une fenêtre de la taille 1, la méthode naïve de mise en correspondance selon pixel met en correspondance chaque pixel individuellement sans faire une addition des résultats intermédiaires des différences. La méthode naïve de mise en correspondance selon pixel est alors la plus rapide. Ce qui est logique, car la mécanique pour partager des valeurs intermédiaires a un coût. Dans le cas de la méthode naïve selon ligne, la synchronisation de ligne en ligne verticalement a un coût supplémentaire.

Pour en savoir plus, le Tableau 4.3 montre les comparaisons de temps d'exécution d'une seule passe de gauche à droite sur les petites images pour le calcul de disparité. La taille des

images traitées est 463×370.

Tableau 4.3 Comparaisons de temps d'exécution (ms) d'une seule passe

Taille de fenêtre	méthode naïve selon pixel	méthode naïve selon ligne	Méthode 2
1	1,88	2,89	2,34
3	5,84	8,69	3,28
5	13,20	18,65	3,53

La différence des temps d'exécution entre la méthode naïve selon pixel et la Méthode 2 pour la taille de fenêtre de comparaison est grande pour certaines tailles de fenêtre. La Méthode 2 met plus de temps dans la mécanique de gestion supplémentaire de fils d'exécution sans avoir l'effet réel pour la fenêtre de comparaison de taille 1.

4.8 Amélioration de la mise en correspondance par bloc avec des techniques

La carte de disparité créée avec la mise en correspondance par bloc a beaucoup de pixels invalides après la vérification de consistance. Pour améliorer la qualité de la carte, nous allons étudier plusieurs techniques potentielles et montrer des méthodes possibles, incluant :

- Filtre bilatéral
- Filtre de soustraction de fond (BilSub)
- Remplissage de disparités des images en plus haute définition
- Enlèvement de taches
- Interpolation efficace

Dans la sous-section 3.3.1, nous avons déjà parlé du filtre bilatéral, nous allons l'employer pour évaluer l'effet. Dans la sous-section 3.3.2, nous avons aussi parlé du filtre de soustraction de fond. Il faut noter que nous avons appliqué la Méthode BilSub sur les images grises avec l'intensité comme élément de calcul au lieu de CIELab suggéré par Tomasi et Manduchi. (voir Tomasi et Manduchi, 1998) Ce choix est pour simplifier la comparaison. Nous pouvons considérer CIELab dans la future.

Le remplissage consiste à donner les valeurs calculées d'une autre méthode aux pixels invalides d'une carte de disparité après la vérification de consistance. Le remplissage sans spécification de détails désigne la recherche à partir d'une plus grande carte de disparité utilisée comme la source de valeur.

Nous allons expliquer en détail ces options. Avant de parler du remplissage de disparités créées avec images en plus haute définition, nous devons préciser quelques questions sur la possibilité et la pertinence.

On peut acquérir des images en plus haute définition, mais l'emploi de ces images a besoin de beaucoup de ressources. Souvent, une carte de disparité de haute définition n'est pas nécessaire, toutefois, une carte de disparité d'une taille raisonnable et assez précise est souhaitable. Les images de haute définition contiennent plus de détails. Il nous intéresse de savoir si nous pouvons profiter de ces informations plus détaillées et créer une carte de disparité dans un délai raisonnable.

4.8.1 Possibilité, méthode et pertinence de remplissage de disparités créées avec images en plus haute définition

Nous avons souvent besoin d'une carte de disparité d'une certaine taille, tandis que nous disposons des images en haute définition. Est-ce possible de se servir d'information des images en haute définition ? Comment en profiter ?

Nous allons montrer la méthode d'amélioration d'une petite carte de disparité avec une plus grande carte de disparité. Nous allons également montrer la possibilité d'employer les images en plus haute définition pour améliorer la qualité de petite carte de disparité.

Méthode pour améliorer une petite carte de disparité avec une plus grande carte de disparité

La Figure 4.7 montre comment remplir une petite carte de disparité avec les valeurs d'une plus grande carte de disparité. Supposons que la plus grande carte de disparité soit $r = 3$ fois plus grande que la petite carte de disparité, nous pouvons calculer un certain produit (par exemple, moyen, médian) de chaque zone correspondante à l'intérieur de la grande carte puis réduire ce produit pour obtenir la valeur à greffer à la plus petite carte.

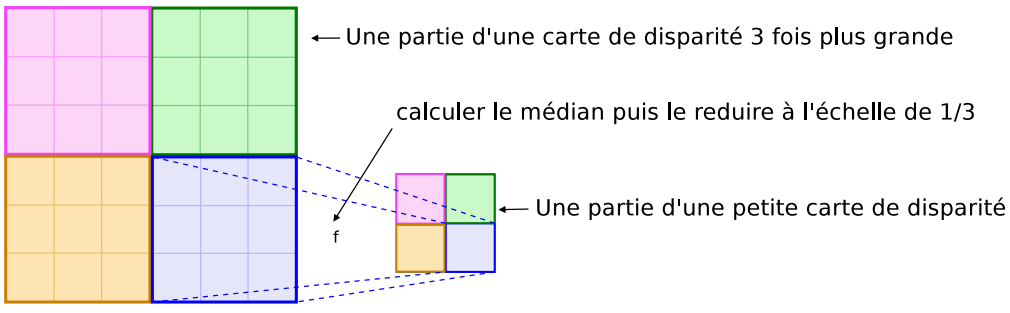


Figure 4.7 Remplissage avec de valeurs d'une plus grande carte de disparité

La méthode employée dans la plupart de nos études est de rechercher la médiane dans une zone carrée qui a une taille équivalant au ratio entre la taille des plus grandes images et celle des plus petites images. Dans nos tests, avec le ratio $r = 3$, nous cherchons la médiane dans

la fenêtre carrée $r \times r$ correspondant à chaque point de la carte de disparité. Une fois trouvé ce médian, nous le divisons par r puis l'arrondissons pour le mettre dans la petite carte de disparité recherchée. Nous pouvons considérer le plancher de la division comme le résultat. Toutefois, après la vérification, nous trouvons que la méthode d'arrondi de la division est préférable. Nous pouvons aussi calculer une plus grande carte de disparité, puis la réduire à l'échelle de $1/r$ pour créer une plus petite carte de disparité. Au lieu du médian, nous pouvons aussi faire le calcul de la moyenne.

Comparaison de précision des méthodes possibles

Le Tableau 4.4 analyse les différentes méthodes simples de création de carte de disparité avec les images de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005), les tailles des images sont 463×370 et 1390×1110. Les tailles de fenêtre de comparaison pour les petites images et les grandes images sont 3 et 9 respectivement en prenant compte du ratio r entre les tailles des grandes images et des petites images. Bien que la précision soit une estimation dans nos études, nous pouvons quand même constater que les images à plus haute définition ont plus de capacité de produire une carte plus précise avec les méthodes simples de la mise en correspondance par bloc. Différentes méthodes produisent un résultat un peu différent, mais la tendance est claire. Les taux d'erreur sont des estimations. L'interpolation MoyenPlusPropagationDuFond est une méthode conservative d'interpolation pour remplir de trous que nous allons introduire plus tard dans la sous-section 4.8.5

Tableau 4.4 Méthodes utilisant la taille de fenêtre 3 sur les petites images de « *Art* »

N°	Méthode	Taux d'erreur pour les pixels visibles	Taux d'erreur globaux
1	Une seule exécution de gauche à droite sur les petites images	41,67%	54,94%
2	Une seule exécution de gauche à droite sur les petites images + interpolation	41,01%	54,23%
3	Réduction à l'échelle de 1/3 avec la médiane des pixels 3×3 correspondants à l'intérieur de la grande carte après la vérification de consistance ; les valeurs invalides seront directement remplies par les pixels de la petite carte	34,60%	49,54%
4	la vérification de consistance sur la petite carte de disparité ; les valeurs invalides seront remplies par le médian réduit à l'échelle de 1/3 des pixels 3×3 de la grande carte avec une seule passe + interpolation	33,29%	48,52%
5	Médian d'une zone de 3×3 sur la petite carte de disparité après la vérification de consistance ; les valeurs invalides seront remplies par le médian réduit à l'échelle de 1/3 des pixels 3×3 de la grande carte avec une seule passe	32,78%	48,19%
6	Réduction à l'échelle de 1/3 avec la médiane des pixels 3×3 correspondants à l'intérieur de la grande carte après la vérification de consistance ; les pixels invalides seront remplis par la médiane de zone 3×3 correspondante de la petite carte	32,61%	47,80%
7	Réduction à l'échelle de 1/3 avec la médiane des pixels 3×3 correspondants à l'intérieur de la grande carte de disparité après une seule exécution de gauche à droite sur les images en plus haute définition	30,96%	46,78%

Les méthodes avec la grande carte après la vérification de consistance prennent plus de temps. Il faut réajuster le temps requis et la précision demandée. C'est une question d'application. Nous employons souvent la combinaison N° 4 parce que c'est un compromis entre la précision et le temps d'exécution. Cette combinaison fait d'abord une vérification de consistance sur les cartes de disparité en employant les petites images (les images standard), les pixels invalides seront remplis avec les disparités réduites à l'échelle de $1/r$ des médians d'une zone $r \times r$ avec les images en plus haute définition d'une taille de r fois plus grande que les petites images (les images standard).

La combinaison N° 3 fait d'abord une réduction d'une carte de disparité après la vérification de consistance en employant les grandes images, les pixels invalides seront remplis par les disparités d'une seule passe de gauche à droite en employant les petites images. La combinaison N° 1 est une simple exécution de la mise en correspondance par bloc de gauche à droite sur les petites images sans d'autres méthodes. Cette méthode est le point de départ de nos études, c'est un choix de compromis entre la qualité de carte de disparité et le temps de traitement requis.

Nous avons calculé une grande carte de disparité avec tous les pixels des images en haute définition au lieu de sélectionner des pixels en question de la carte de disparité de taille standard après la vérification de consistance pour en chercher des valeurs de la grande carte de disparité. La raison tient au fait qu'il y a beaucoup de disparités invalides, selon nos études sur les images de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005), 37,01% sont visibles, mais invalidées si nous faisons une vérification de consistance sur les cartes de disparité obtenues avec une fenêtre de taille 5. Si nous faisons un remplissage de disparités en employant les images en plus haute définition avec une fenêtre de taille $5r = 15$ après la vérification de consistance pour être considérées comme valides, 29,10% de disparités des pixels potentiels seront remplacés, ou 10,77% de tous les pixels visibles de la carte de disparité. L'algorithme que nous avons employé est une accumulation et un réajustement à chaque étape, il sera plus coûteux de faire une approche sélective pour un tel pourcentage de pixels invalides.

4.8.2 Remplissage avec de disparités créées avec images en plus haute définition

Pour une correspondance pas assez fiable, la vérification de consistance invalide beaucoup de disparités, nous avons intérêt à rechercher les valeurs de la carte de disparité des images en plus haute définition, parce que nous supposons que les images en plus haute définition contiennent plus d'information. La taille de la fenêtre de comparaison des images en plus haute définition est r fois celle des petites images utilisées dans nos études. Pour éviter que les valeurs recherchées soient encore invalides dans la plus grande carte de disparité, nous

calculons dans un premier temps une seule exécution de gauche à droite sans la vérification de consistance sur la plus grande carte de disparité.

Nous pouvons aussi faire une plus grande carte de disparité avec la vérification de consistance pour ensuite procéder au remplissage. Si la carte recherchée a encore de points invalides, nous tenterons de les combler avec des images en encore plus grande définition, ou avec une différente taille de fenêtre de comparaison, ou encore avec une autre méthode de remplissage, et même avec l'interpolation.

4.8.3 Enlèvement de taches

Nous considérons que les petites zones isolées ayant des disparités très différentes du voisinage sont des taches (*speckles*). L'option enlèvement de taches est une option qui cherche à éliminer des valeurs qui ont beaucoup changé dans une petite zone. Nous supposons que les disparités changent avec une certaine continuité. Une petite zone ayant trop de changement peut être due au bruit, à la mauvaise correspondance ou le point n'est pas visible pour un appareil photo. De différentes implémentations ont réalisé ce type d'enlèvement de taches, par exemple, OpenCV (voir Bradski et Kaehler, 2008, p. 443).

4.8.4 Interpolation Moyen

La méthode Moyen est une option qui remplit des valeurs invalides par interpolation un peu comme ce que fait le remplissage. Cependant, cette option essaie de calculer la moyenne autour de chaque pixel lorsque la différence des valeurs de ces pixels est sous un certain seuil. Ensuite, si le pixel à droite du pixel considéré est invalide, il est rempli avec cette moyenne. Cette méthode suppose que la disparité présente une certaine continuité.

L'option Moyen présente certains problèmes. Cette méthode ne prend pas en compte les particularités des images, par exemple, le fait que le changement d'intensité à la bordure soit normalement grand parce que seulement ces différences d'intensité rendent visibles les bordures d'objet. Nous avons amélioré l'option Moyen pour créer une autre option MoyenPlusPropagationDuFond qui prend en compte plus d'information, et devient donc plus précise.

4.8.5 Interpolation MoyenPlusPropagationDuFond

La méthode MoyenPlusPropagationDuFond cherche des valeurs sous certaines conditions définies par les particularités des images. Les travaux de Birchfield et Tomasi (voir Birchfield et Tomasi, 1998, p. 5) et de Hirschmüller (voir Hirschmüller, 2003, pp. 34–35) nous ont inspiré pour proposer cette méthode un peu conservative dans le sens que cette méthode ne fait l'interpolation que si certaine condition prédéfinie est satisfaite.

Il faut employer toutes les informations disponibles pour se rapprocher au plus près de la carte idéale. Si la différence d'intensité entre deux pixels sous un certain seuil varie un peu, il faut prendre la moyenne de cette petite zone et la propager vers la droite dans une exécution de gauche à droite. Sinon, si la variation d'intensité est grande, nous supposons que cette zone se trouve soit à la bordure d'objet, soit qu'elle est invisible par un appareil photo. Cette méthode simple fait la distinction à cette étape au lieu de celle de la vérification de consistance pour éviter la sauvegarde d'information supplémentaire. Cette méthode prend simplement la moyenne d'une petite zone pour propager l'information en supposant que la disparité a la tendance de garder sa valeur dans une petite zone. Nous pouvons aussi calculer une fonction linéaire en supposant que la disparité change en continuité dans une direction, par exemple, de gauche à droite.

Il peut y avoir des zones sans valeur valide. Pour améliorer la situation, nous recherchons la plus petite disparité autour du pixel étudié dans une zone d'une taille définie, ensuite nous la remplissons avec la plus petite des disparités autour du pixel en supposant que ce sera la valeur manquante. Souvent, les zones des bordures présentent des problèmes dus à un grand changement d'intensité. Parce que les bordures se trouvent entre les objets, on suppose que le fond de la scène est manquant. Nous utilisons alors la valeur minimale du voisinage d'un pixel pour combler la carte de disparité par simplicité. Nous pouvons éventuellement segmenter les images pour mieux interpoler la carte. La contrainte est évidente : il faut avoir au moins une certaine densité de disparité valide pour que la supposition de manquement de fond tienne.

Le principe de cette méthode est connu, mais la méthode est nouvelle dans le sens que cette méthode simple et conservative ayant une bonne performance a été décrite dans le cas de pixel sans segmentation pour la première fois à ma connaissance.

Algorithme de la méthode MoyenPlusPropagationDuFond

Nous proposons une approche détaillée de l'algorithme utilisé pour la méthode MoyenPlusPropagationDuFond en deux étapes.

D'abord, définissons un seuil de différence d'intensité des images étudiées S_{img} et un seuil de différence de disparité S_{disp}. Dans nos études, nous avons $S_{\text{img}} = 5$ et $S_{\text{disp}} = 3$.

Première étape

Pour chaque ligne de la carte de disparité :

1. Nous prenons une valeur de disparité d_{ik} jusqu'à ce que cette valeur soit valide, c'est-à-dire une valeur plus grande que 0, où i est l'indice de cette valeur dans la ligne. Si la différence absolue entre une disparité v et chacun de ses voisins directs dans les quatre

directions est moins d'un seuil de propagation de disparité S_v, nous additionnons ces différences pour obtenir la moyenne v_m.

2. Si $i+1 \geq d_{ik}$, nous calculons la différence absolue d'intensité d_{i+1} entre imgG_{i+1} et $\text{imgD}_{i+1-d_{ik}}$. Les symboles imgG et imgD correspondent respectivement à l'image de gauche et à l'image de droite.
3. Si $d_{i+1} < S_{\text{img}}$, nous allons propager v_m à droite à la condition que $d_{i+j}(j > 0)$ soit 0.

Pour plus de clarté, nous pouvons reprendre ce qui précède comme suit :

Si d_{ik} est la première disparité valide, nous cherchons les plus proches disparités valides en haut d_{ih}, en bas d_{ib}, à gauche d_{gk} et à droite d_{dk} du d_{ik}. Si la différence absolue entre d_{ik} et ses voisins est moins de S_{disp}, nous calculons la moyenne de ces disparités avec l'équation 4.3.

$$d_v = \frac{d_{ik} + \sum_{j\in\{h,b\}} d_{ij} + \sum_{j\in\{g,d\}} d_{jk}}{n}, \text{si abs}(d_{ik} - d_{ij}) < S_{\text{disp}} \text{ où}(j \in \{h,b\}) \text{ et si abs}(d_{ik} - d_{jk}) < S_{\text{disp}} \text{ où}(j \in \{g,d\}), \; n \text{ est le nombre de pixels qui satisfont les conditions ci-dessus plus 1.} \tag{4.3}$$

Pour chaque d_{lk} invalide à droite de d_{ik} jusqu'à la prochaine disparité valide,

$$d_{lk} = d_v, \text{ si abs}(\text{imgG}_{lk} - \text{imgD}_{l-d_{ik}}) < S_{\text{img}}, \text{où } l \geq d_{ik} \tag{4.4}$$

La première étape de cette méthode est inspirée de la méthode introduite par Birchfield et Tomasi (voir Birchfield et Tomasi, 1998, p. 5).

Après cette étape, il y a encore des zones vides. Nous passons à l'étape suivante.

Deuxième étape

Nous définissons un seuil $S_{\text{Himg}} = \text{plancher}(S_{\text{img}}/2)$ et une valeur de disparité maximale $d_{\max}$.

Pour chaque ligne k de la carte de disparité, si un pixel est invalide :

1. Pour chaque pixel invalide d_i de la carte de disparité, nous cherchons parmi les disparités valides et plus petites que $d_{\max}$ dans une fenêtre centrée sur le pixel d_i à trouver la valeur minimale $d_{\min}$. La fenêtre est définie respectivement par la gamme $[i - S_{\text{Himg}}, i + S_{\text{Himg}}]$ et la gamme $[k - S_{\text{Himg}}, k + S_{\text{Himg}}]$ horizontalement et verticalement.
2. Propager $d_{\min}$ à droite.

Pour encore plus de clarté, nous pouvons aussi poursuivre comme suit :

Pour chaque d_{lk} invalide, nous calculons d_v et la propageons à droite :

$$\begin{aligned} d_{lk} &= d_v; \\ d_v &= \min(d_{ij}), i \in [i - S_{\text{Himg}}, i + S_{\text{Himg}}], j \in [k - S_{\text{Himg}}, k + S_{\text{Himg}}], \\ & \quad d_{ij} \text{ est valide.} \end{aligned} \tag{4.5}$$

La deuxième étape de cette méthode est inspirée par la méthode introduite dans la sous-section 3.14.2, sauf que nous n'avons pas fait la segmentation. (voir Hirschmüller, 2003, pp. 34–35)

La Figure 4.8 illustre dans une certaine mesure l'interpolation MoyenPlusPropagationDuFond.

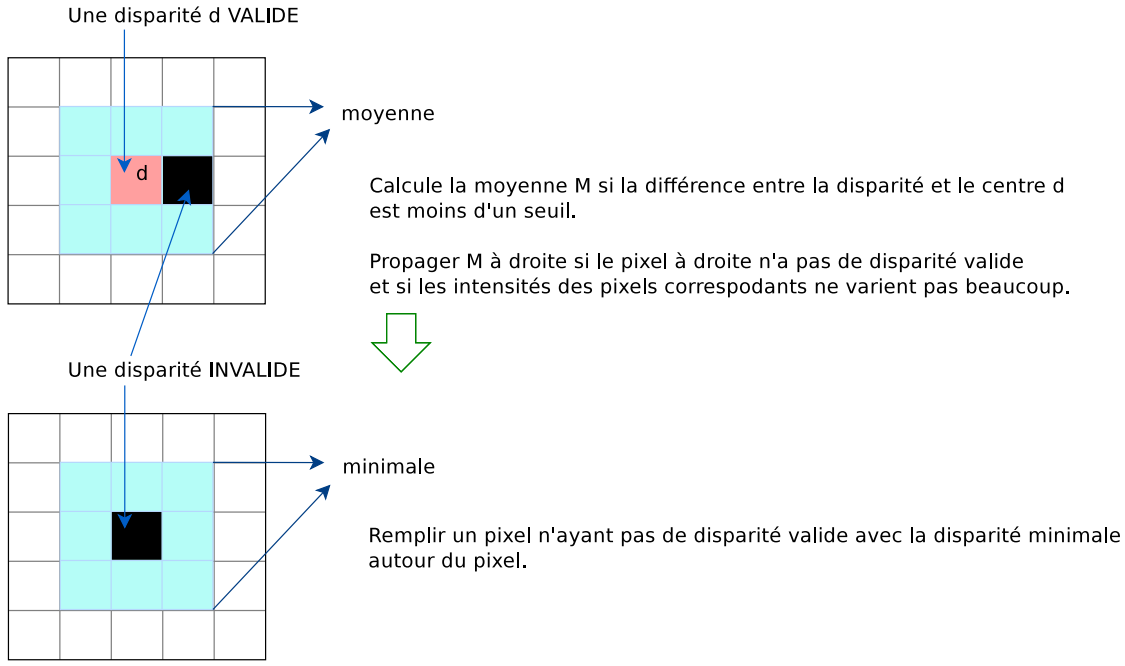


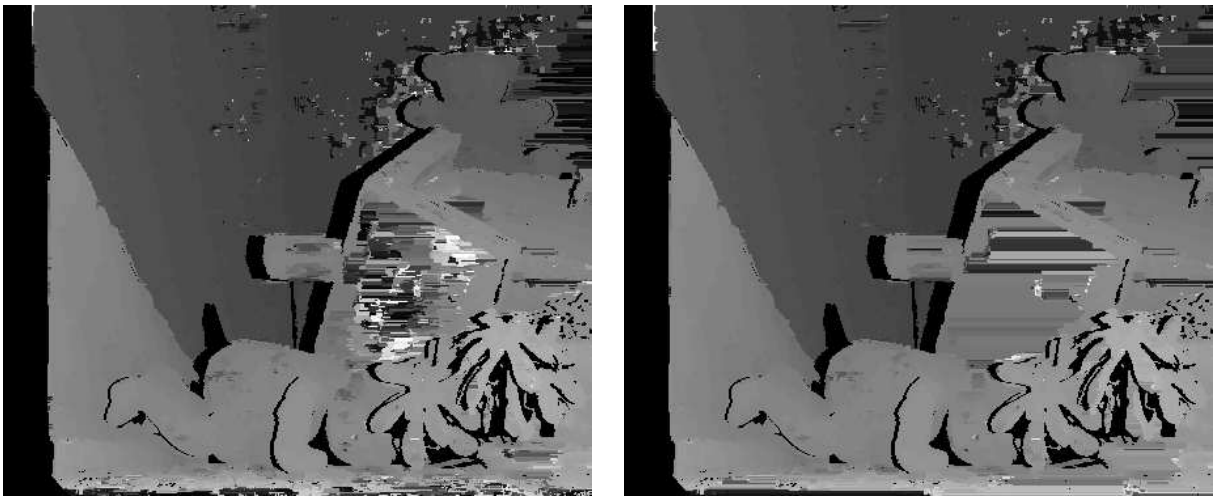
Figure 4.8 Interpolation MoyenPlusPropagationDuFond

4.9 Effets des techniques

Nous avons fait des expériences pour évaluer les techniques potentielles que nous avons introduites. Les expériences suivantes ont pour but de montrer l'efficacité de chaque technique. Certaines sont pour but d'avoir une idée claire de nos expériences. La plupart des techniques ici ne sont pas nouvelles. Nous avons déjà mentionné la BilSub dans sous-section 3.3.2 de la revue de littérature.

4.9.1 Effet de l'enlèvement de taches

Des petites régions variant beaucoup du voisinage sont souvent considérées comme des taches, dues à la difficulté de correspondance, aux bruits, au manque de texture ou à la réflexion. Nous allons étudier l'effet de l'enlèvement de taches. La Figure 4.9 compare l'effet de l'enlèvement de taches. Les images traitées sont les petites images de taille 450×375 de « *Teddy* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005). Les cartes de disparité sont obtenues avec une vérification de consistance sur les petites images puis un enlèvement de taches ou sans enlèvement de taches. Pour combler les pixels invalides laissés par la vérification de consistance ou l'enlèvement de taches, nous procédons une interpolation MoyenPlusPropagationDuFond introduite dans la sous-section 4.8.5. Nous avons enlevé les pixels invisibles en les mettant la valeur 0.



sans l'enlèvement de taches, taille : 5,
taux d'erreur des pixels visibles : 20,92%

avec l'enlèvement de taches, taille : 5,
taux d'erreur des pixels visibles : 19,44%

Figure 4.9 Comparaison des cartes obtenues selon méthode avec les images de « *Teddy* »

Nous pouvons voir que la technique enlèvement de tache peut bel et bien lisser la carte, sauf que l'on doit porter une attention particulière quant aux paramètres et la méthode employée pour ne pas enlever des pixels importants pour l'interpolation. Évidemment, on peut considérer une bonne combinaison de l'enlèvement de taches et de l'interpolation pour arriver à un résultat satisfaisant.

Pourtant, les chiffres de taux d'erreur des pixels visibles dans la Figure 4.10 montrent que pour les images de « *Teddy* », la différence entre avec ou sans l'enlèvement de taches est limitée. Une autre raison est que notre méthode est très élémentaire.

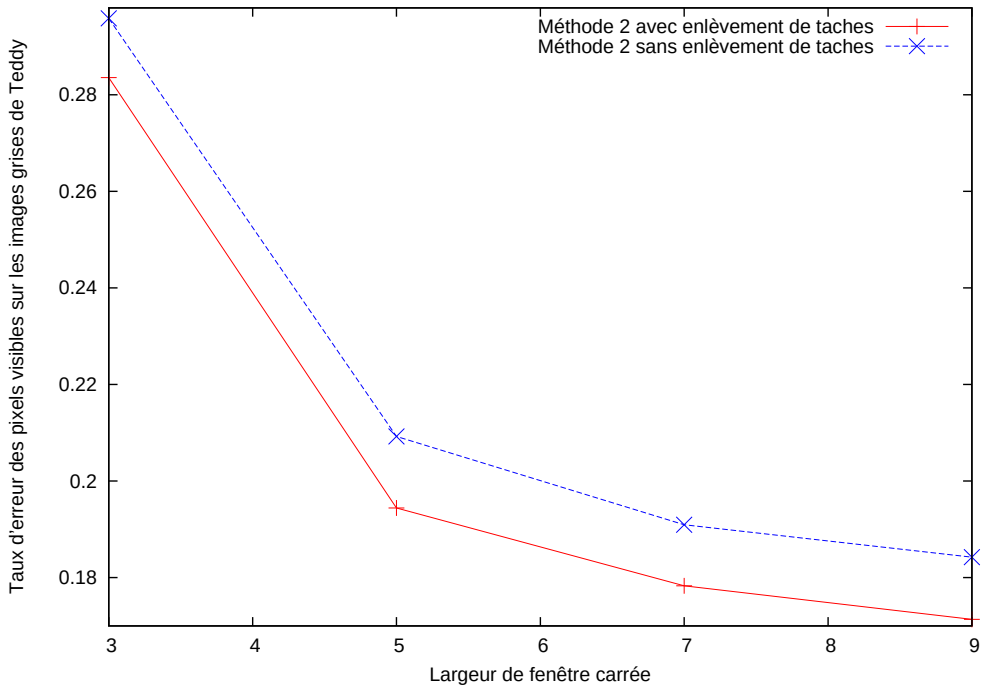
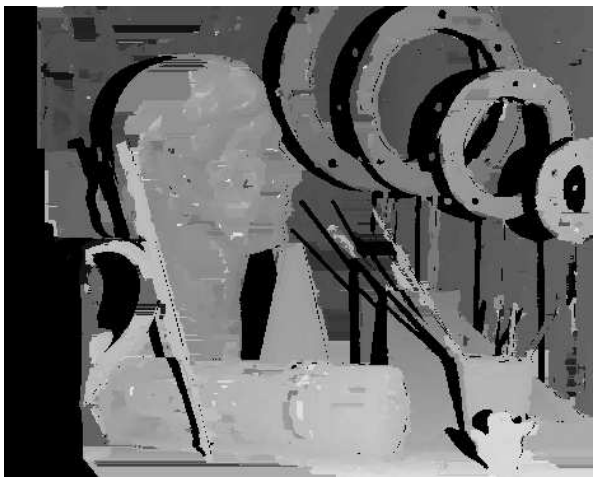


Figure 4.10 Taux d'erreur des pixels visibles en fonction de l'enlèvement de taches sous la Méthode 2 et l'interpolation sur les images de « *Teddy* »

Pour confirmer notre conclusion, la Figure 4.11 montre les cartes de disparité traitées avec ou sans l'enlèvement de taches avec les petites images de taille 463×375 de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005). Nous pouvons voir que la différence est minimale, pourtant visible, surtout aux régions de petites variations.

sans l'enlèvement de taches, taille : 5,
taux d'erreur des pixels visibles : 32,17%



avec l'enlèvement de taches, taille : 5,
taux d'erreur des pixels visibles : 30,94%

Figure 4.11 Comparaison des cartes obtenues selon méthode avec les images de « *Art* »

La Figure 4.12 montre la comparaison en employant la Méthode 2 avec ou sans l'enlèvement de taches, montre que l'enlèvement de taches peut diminuer un peu le taux d'erreur avec le remplissage de disparités des plus grandes images. Les études se font avec le groupe de paramètres tels que définis précédemment et avec la simple mise en correspondance par bloc sans le remplissage de disparités des images en plus haute définition. Dans les tests, la disparité maximale est 80, le seuil de vérification est 0, et avec ou sans remplissage de disparités d'une seule passe sur les images en plus haute définition. Nous pouvons voir que l'enlèvement de taches peut aider à diminuer le taux d'erreur en employant une bonne combinaison de paramètres et de remplissage.

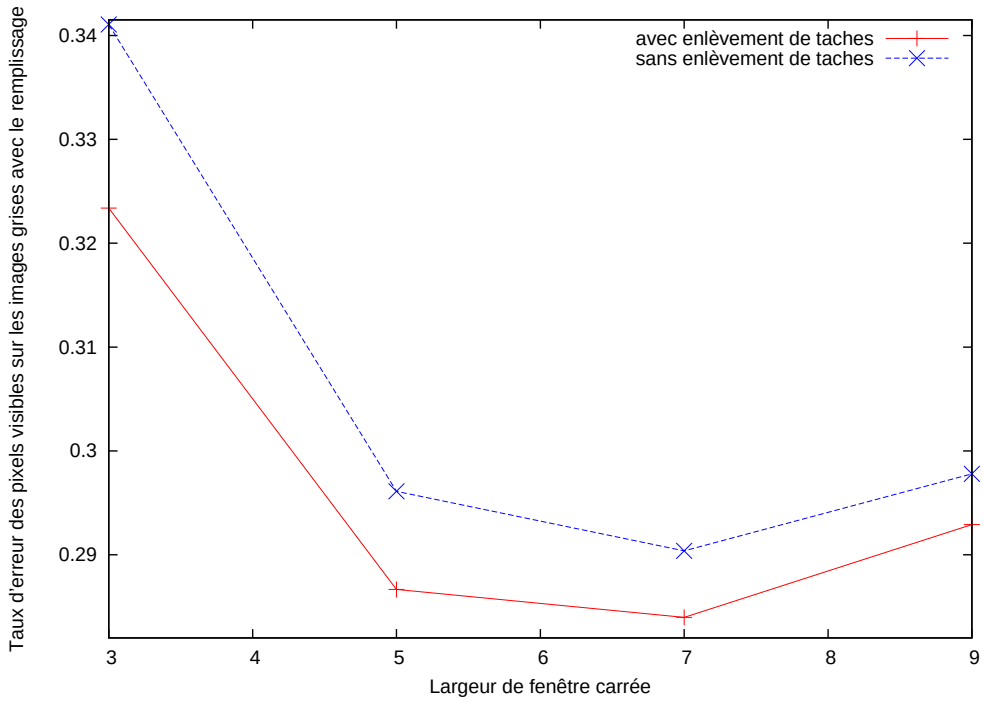


Figure 4.12 Taux d'erreur des pixels visibles en fonction de l'enlèvement de taches sur les images de « *Art* »

Si l'enlèvement de taches est combiné avec le remplissage de disparités des images en plus haute définition comme dans la Figure 4.12, le résultat sera meilleur. La Figure 4.12 montre aussi que l'enlèvement de taches, lorsque nous employons le groupe de paramètres tels que définis précédemment pour le remplissage de disparités en employant les images en plus haute définition, peut augmenter la précision.

Nous devons donc employer l'enlèvement de taches avec prudence, en mesurant le résultat obtenu. Dans nos études, les résultats de la technique d'enlèvement de taches sont limités à nos échantillons. Cependant, nos estimations peuvent constituer un bon indice.

4.9.2 Effet de la Méthode BilSub

Dans la sous-section 3.3.2, nous avons parlé de la Méthode BilSub. Cette Méthode permet d'éliminer le fond avec l'équation 3.13. Nous avons calculé le taux d'erreur avec la technique enlèvement de tache et la technique remplissage de disparité des images en plus haute définition. Le résultat montre une différence constante. La soustraction de BilSub s'applique sur les grandes images avec un certain groupe de paramètres. Les paramètres doivent varier selon la taille de l'image ou le type d'image, nous avons appliqué le même groupe de paramètres pour simplifier les études. Les images employées sont les images de « *Art* » et

de « *Laundry* » de taille de 463 × 370 et de taille 1390 × 1110 de Middlebury présentées par Blasiak *et al.*. (voir Blasiak *et al.*, 2005) Le résultat avec ce certain groupe de paramètres est montré dans la Figure 4.13. Nous pouvons voir que la Méthode BilSub peut améliorer la précision, mais de façon limitée.

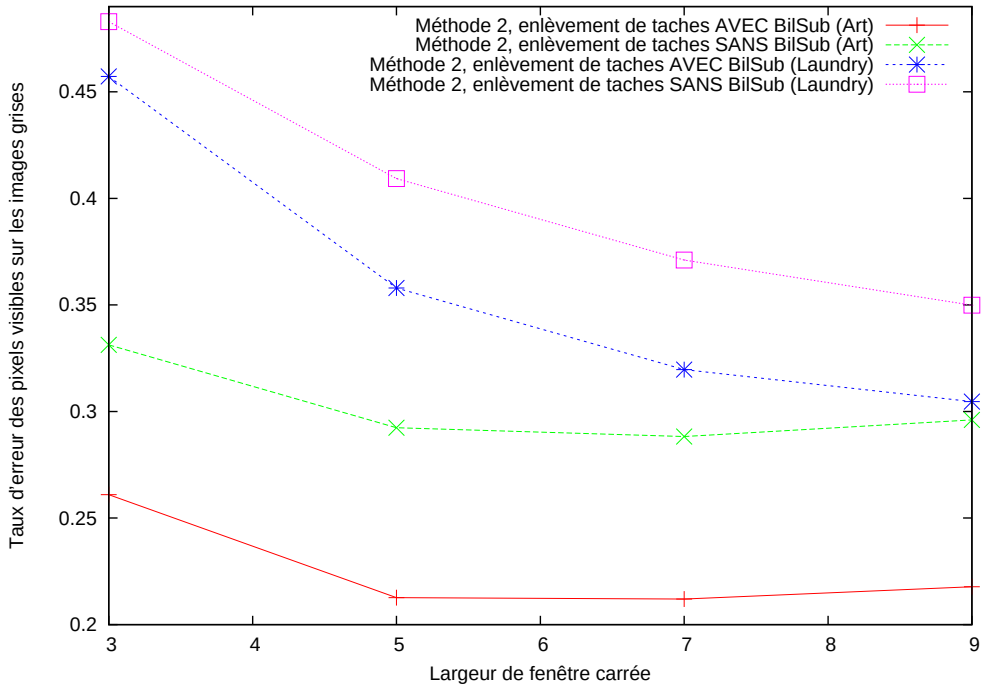
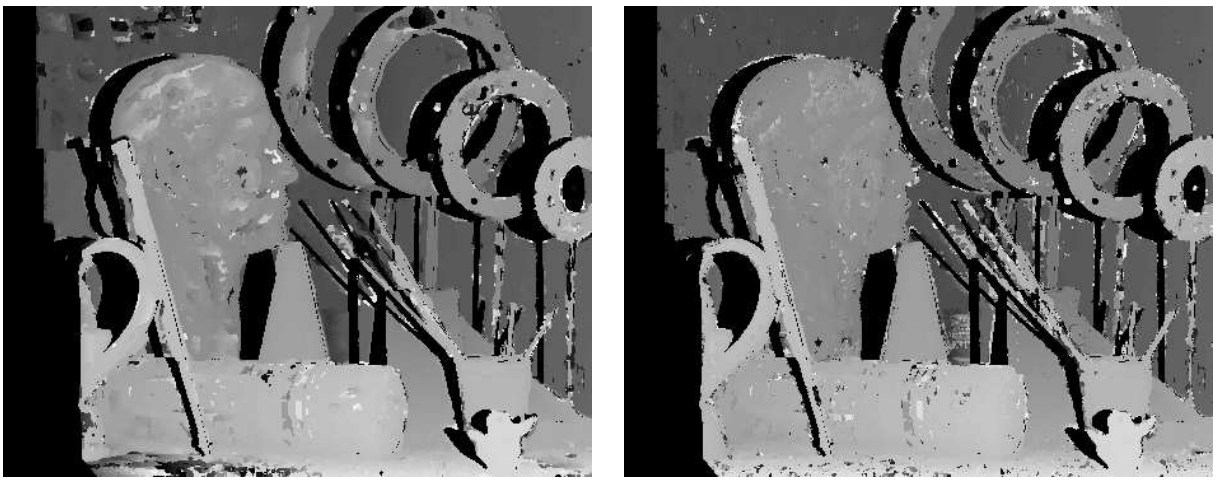


Figure 4.13 Taux d'erreur des pixels visibles selon BilSub avec le remplissage, l'enlèvement de taches et la Méthode 2

La Figure 4.14 montre les cartes de disparité obtenue des images de « *Art* » de Middlebury en employant la technique d'enlèvement de taches, la technique remplissage de disparité des images en plus haute définition et avec ou sans la Méthode BilSub. Les images de « *Art* » de Middlebury sont présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005). La taille de la fenêtre de comparaison employée est 5. La carte de disparité a été augmentée à l'échelle de 3 afin de l'afficher clairement.



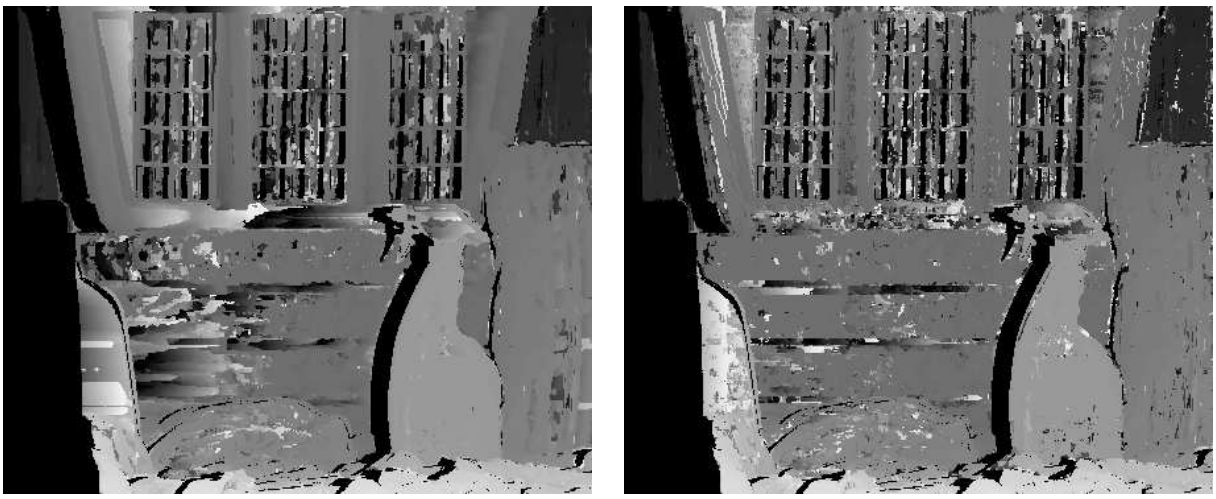
sans BilSub, taille : 5,
taux d'erreur des pixels visibles : 29,61%

avec BilSub, taille : 5,
taux d'erreur des pixels visibles : 22,78%

Figure 4.14 Carte obtenue avec enlèvement de taches et la Méthode 2, sans ou avec BilSub pour les images de « *Art* »

Nous pouvons constater la BilSub peut lisser certaines zones de beaucoup de variations. Mais l'amélioration n'est pas très visible. Pour connaître plus l'effet, nous avons testé la BilSub avec les images de *Laundry* de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005).

La Figure 4.15 montre les cartes de disparité obtenue des images de *Laundry* en employant la technique d'enlèvement de taches, la technique remplissage de disparité des images en plus haute définition et avec ou sans la Méthode BilSub. La taille de la fenêtre de comparaison employée est 5. La carte de disparité a été augmentée à l'échelle de 3 afin de l'afficher clairement.



sans BilSub, taille : 5,
taux d'erreur des pixels visibles : 41,45%

avec BilSub, taille : 5,
taux d'erreur des pixels visibles : 37,16%

Figure 4.15 Carte obtenue avec enlèvement de taches et la Méthode 2, sans ou avec BilSub pour les images de *Laundry*

Nous pouvons constater que dans certaines zones de haut contraste, la BilSub peut aider à faire la mise en correspondance. En même temps, elle a aussi introduit des bruits.

Le filtre bilatéral a un coût. Le temps varie selon les paramètres et les images traitées. Dans nos études, pour deux images de « *Art* » de taille 1390×1110 sur Intel® Core™ i7-2600K @ 3,40 GHz sous Windows 7 Professional sous un certain groupe de paramètres, le temps d'exécution est 1,65 s en moyenne. Si nous appliquons le filtre avec un autre groupe de paramètres presque sans changer les taux d'erreur, le temps d'exécution est 0,57 s en moyenne. Bien sûr cette partie peut être implémentée sur GPGPU afin d'accélérer le traitement si nécessaire. La soustraction de BilSub peut seulement s'appliquer sur les images en plus haute définition parce que normalement les images en plus haute définition ont plus de détails et par conséquent, plus de variation locale.

La Figure 4.16 étudie l'effet d'enlèvement de taches avec la technique BilSub et la technique remplissage de disparité des images en plus haute définition. Nous pouvons voir que dans ce cas, l'effet d'enlèvement de taches est toujours positif comme confirmé dans la sous-section 4.9.1.

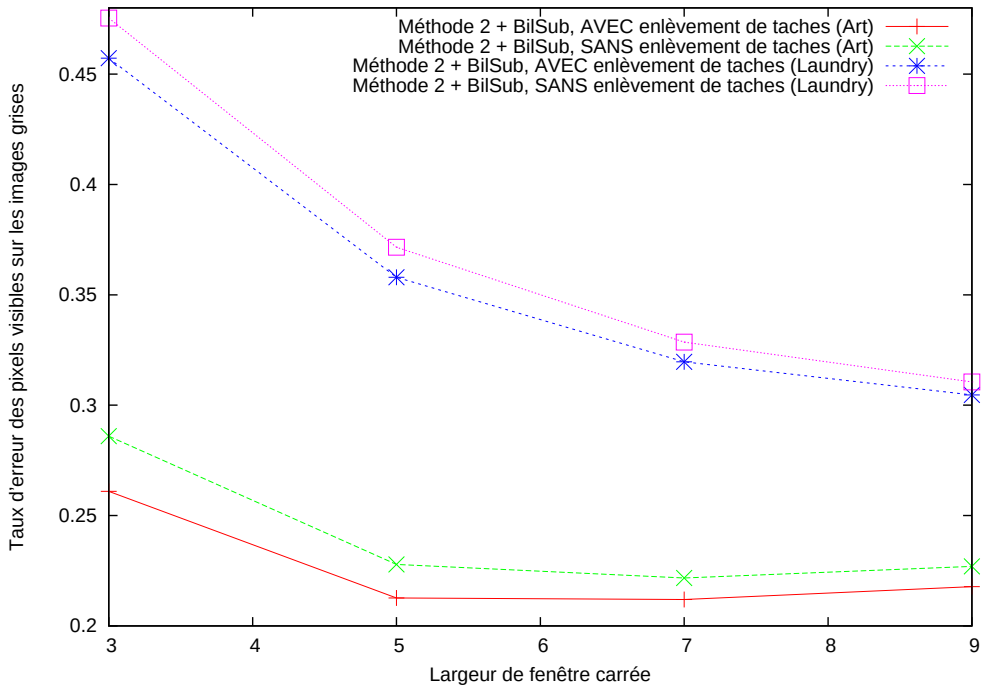


Figure 4.16 Taux d'erreur des pixels visibles selon enlèvement de taches avec BilSub, le remplissage et la Méthode 2

4.9.3 Effet du remplissage

La Figure 4.17 montre les taux d'erreur des pixels visibles calculés avec la technique enlèvement de tache, et avec la technique remplissage de disparité des images en plus haute définition, ou sans la technique remplissage, avec l'interpolation affiche une différence constante, ou encore une seule passe de gauche à droite sans le remplissage ni l'interpolation. Le remplissage ou l'interpolation sont pour but de remplir des pixels invalides après la vérification de consistance. La technique remplissage peut améliorer la performance, mais seulement dans une certaine mesure.

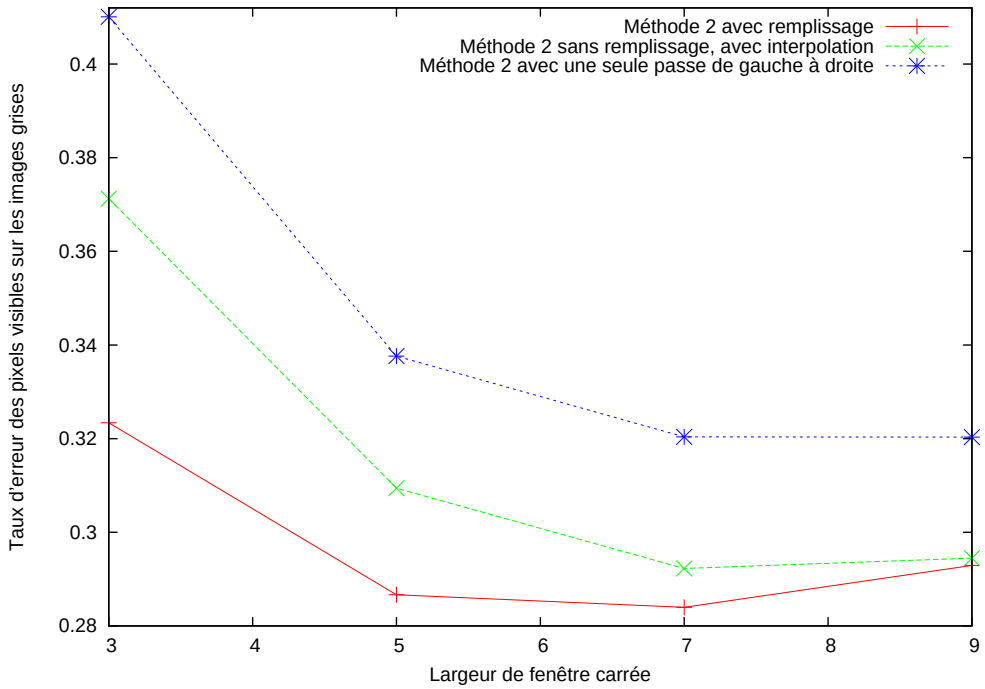


Figure 4.17 Taux d'erreur des pixels visibles en fonction du remplissage ou de l'interpolation avec la Méthode 2

Les images traitées sont les petites et les grandes images de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005), les tailles sont 463×370 et 1390×1110. Nous avons besoin de deux types de taille d'image pour mieux faire le remplissage.

Si nous doutons du résultat pour d'autres types d'images, nous avons essayé plus d'images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Les images de 2005 employées incluent les plus grandes et les plus petites de « *Art* », de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». Pour ce qui est des images de 2003, elles incluent les images les plus grandes et les plus petites de « *Cones* » et de « *Teddy* ». La disparité maximale des petites images de 2005 est 79, celle des petites images de 2003 est 63. La disparité maximale pour les grandes images de 2005 et de 2003 est 239 et 255 respectivement. Les ratios de taille entre les grandes images et les petites images de 2005 et de 2003 sont 3 et 4 respectivement. La Figure 4.18 montre que la moyenne des taux d'erreur des pixels visibles calculés avec la technique enlèvement de tache, et avec la technique remplissage de disparité des images en plus haute définition, ou sans la technique remplissage, avec l'interpolation affiche une différence constante, ou encore une seule passe de gauche à droite sans le remplissage ni l'interpolation.

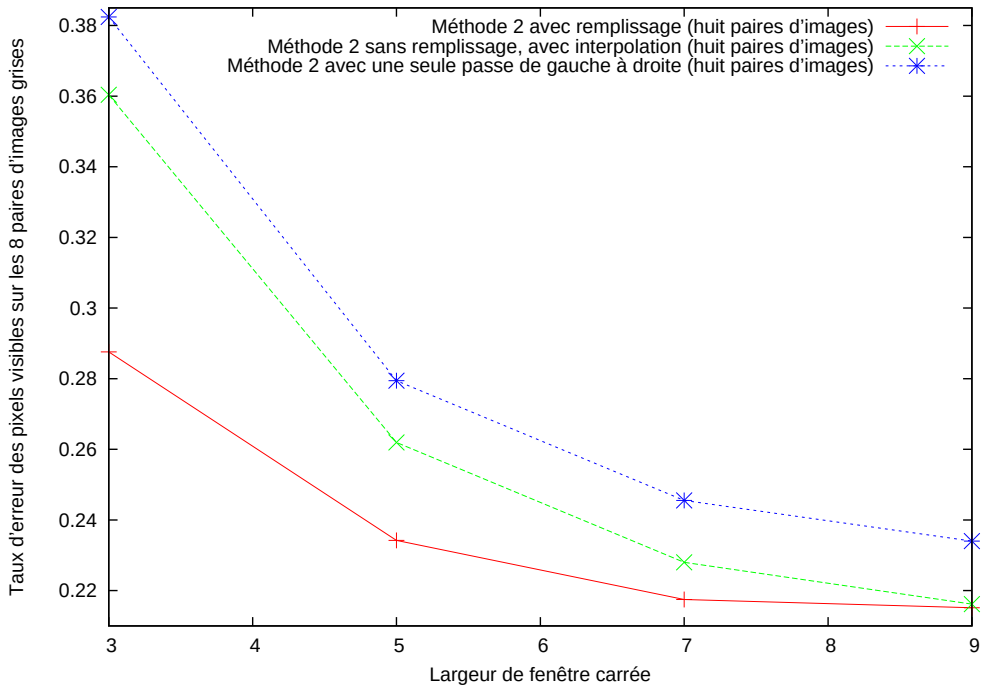


Figure 4.18 Moyennes des taux d'erreur des pixels visibles en fonction du remplissage ou de l'interpolation avec la Méthode 2 pour 8 paires d'images

Le résultat avec le remplissage appliqué aux images en plus haute définition est meilleur que celui produit sans le remplissage. Parce qu'une image en plus haute définition contient plus de détail, nous pouvons espérer voir une amélioration.

Dans nos études, nous avons employé 0 comme seuil de vérification de consistance. Le remplissage diminue le taux d'erreur. La taille de la fenêtre de comparaison par convention est proportionnelle à celle des images plus petites selon le ratio r. Nous pensons que la carte de disparité créée avec les images en plus haute définition peut être plus proche de la réalité. La Figure 4.18 de la sous-section 4.9.3 confirme cette hypothèse.

4.9.4 Effet de l'interpolation MoyenPlusPropagationDuFond

Nous avons présenté dans la Figure 4.19 la comparaison entre l'interpolation, le remplissage et diverses combinaisons. Le remplissage correspond au remplissage de disparités des images en plus haute définition. L'interpolation fait référence à la Méthode MoyenPlusPropagationDuFond qui prend en compte des particularités des images. Pour ce qui est de la vérification, il s'agit de la vérification de consistance. Concrètement, Le cas N° 1 passe la vérification de consistance sur les grandes et les petites cartes de disparité, puis fait le remplissage et l'interpolation. Le cas N° 2 fait la vérification de consistance sur les petites cartes

de disparité, puis fait le remplissage de disparités obtenues après une seule passe de gauche à droite sur les grandes images. Le cas N° 3 fait la vérification de consistance sur les petites cartes de disparité puis passe l'interpolation MoyenPlusPropagationDuFond. Finalement, le cas N° 4 emploie les grandes images pour créer une carte de disparité, puis on la réduit à l'échelle de $1/r$, ensuite passe l'interpolation MoyenPlusPropagationDuFond pour avoir une carte de disparité.

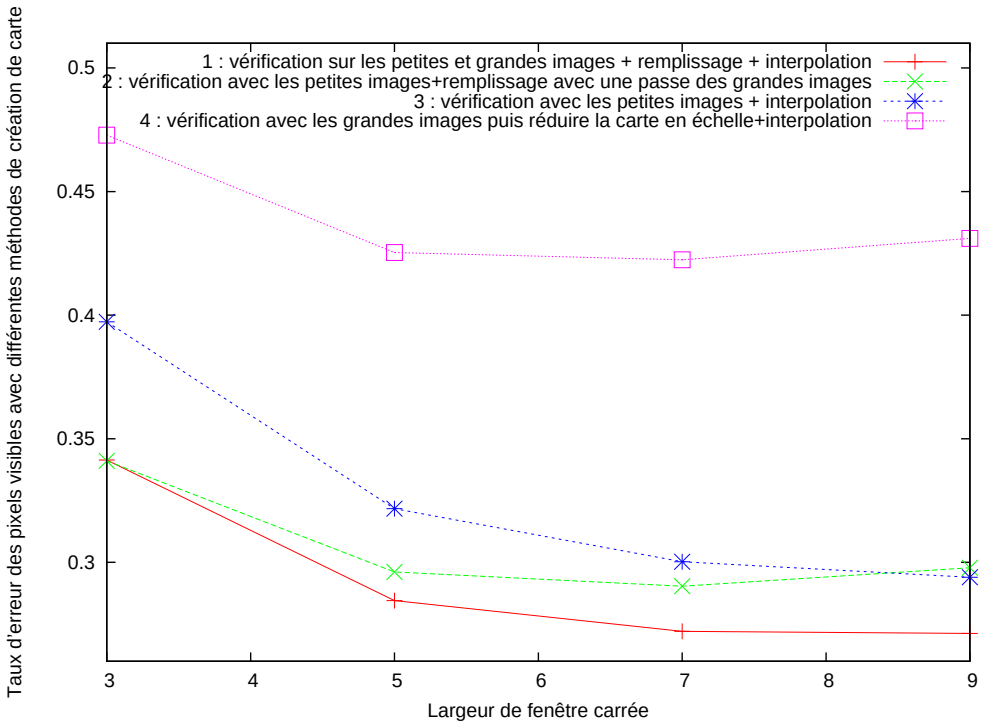


Figure 4.19 Comparaisons des taux d'erreur avec des combinaisons de techniques avec les images de « *Art* »

Nous allons montrer le résultat pour plus d'images dans la Figure 4.23 plus loin.

Une bonne interpolation peut très bien compléter les autres techniques que nous avons mentionnées plus haut. Nous pouvons voir qu'une fois l'interpolation activée, nous avons une meilleure précision. Une bonne combinaison des techniques peut encore augmenter la précision.

La Figure 4.20 compare une carte de disparité après la vérification de consistance et une carte après remplissage et interpolation avec MoyenPlusPropagationDuFond. La taille de la fenêtre utilisée est 5. Ce remplissage consiste à remplir la petite carte de disparité avec une grande carte de disparité après la vérification de consistance ou une seule passe de gauche à droite sans vérification de consistance.

N° 1 : vérification sur les petites et grandes images + remplissage + interpolation

N° 2 : vérification de consistance avec les petites images + remplissage avec une passe de HD

N° 3 : vérification de consistance avec les petites images + interpolation

N° 4 : vérification avec les grandes images, réduire la carte et interpolation

Figure 4.20 Comparaisons des cartes obtenues selon les combinaisons avec les images de « *Art* »

Dans la Figure 4.19 et la Figure 4.20, lorsque nous écrivons « avec les petites et grandes images », nous voulons dire que les actions ont été exécutées sur les grandes et les petites images.

Nous pouvons constater que pour les images de « *Art* », une interpolation combinée avec les grandes images du cas N° 1 peut légèrement surpasser les cas N° 2, N° 3 et N° 4 quant à la performance.

La qualité de carte de disparité dépend de type d'image, par exemple, la Figure 4.21 montre que pour les images de « *Laundry* », les cas N° 1 et N° 2 produisent le taux d'erreur

similaire. Les cas N° 3 et N° 4 ont presque le même taux d'erreur. La Figure 4.22 montre que pour les images de « *Dolls* », la tendance du cas N° 4 est similaire à celle des images de « *Art* ».

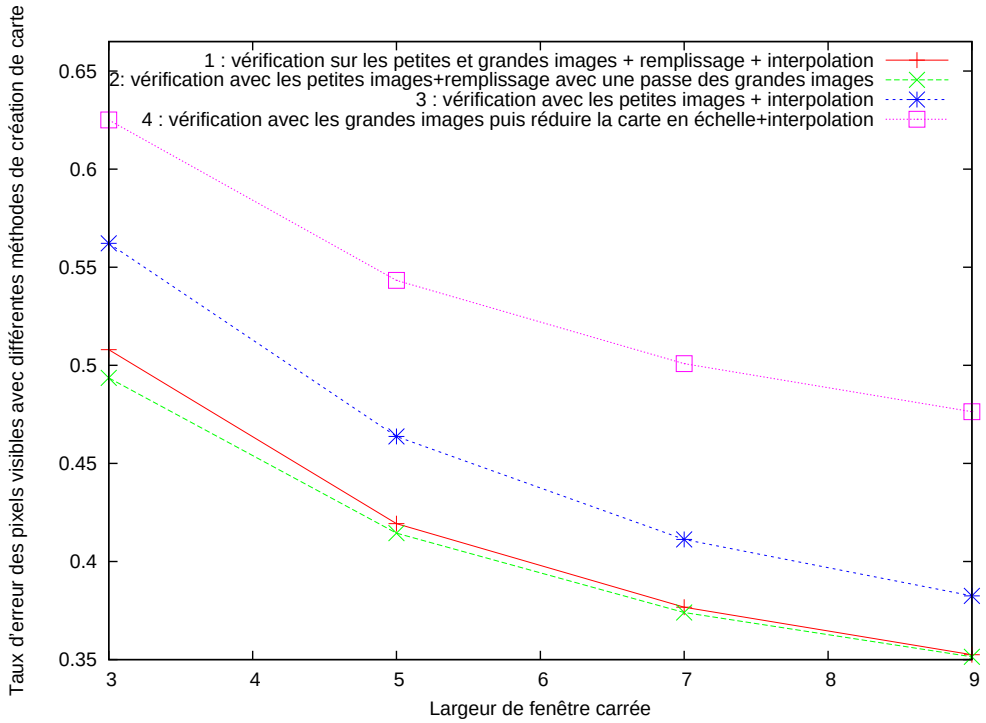


Figure 4.21 Comparaisons des taux d'erreur avec des combinaisons de techniques avec les images de « *Laundry* »

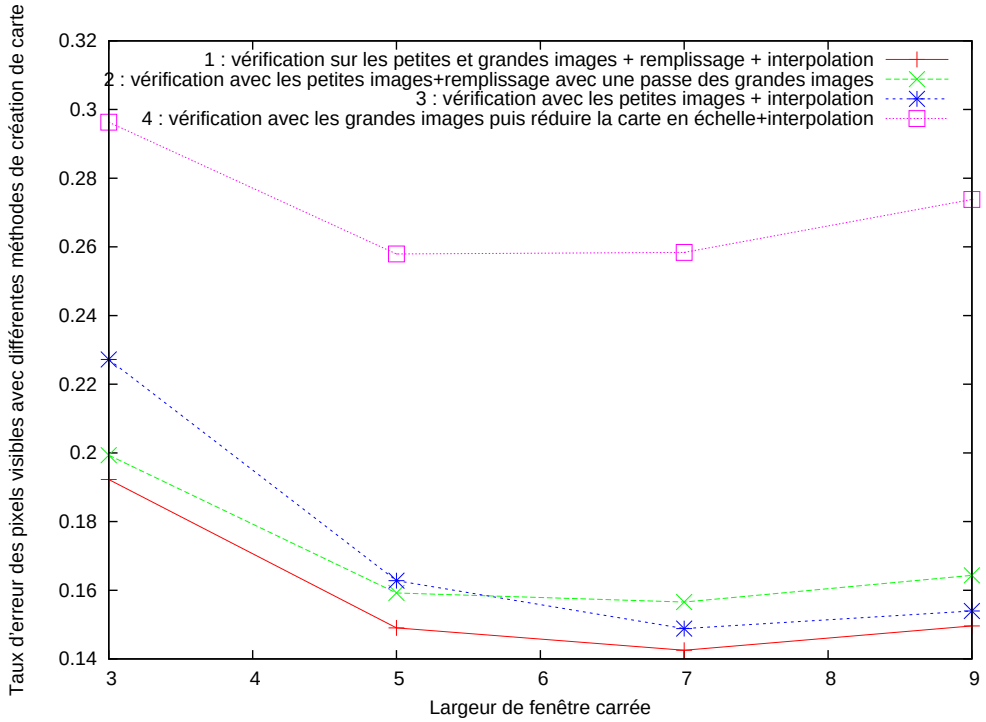


Figure 4.22 Comparaisons des taux d'erreur avec des combinaisons de techniques avec les images de « *Dolls* »

Nous avons essayé plus d'images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Les images de 2005 employées incluent les plus grandes et les plus petites de « *Art* », de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». Pour ce qui est des images de 2003, elles incluent les images les plus grandes et les plus petites de « *Cones* » et de « *Teddy* ». La disparité maximale des petites images de 2005 est 79, celle des petites images de 2003 est 63. Les disparités maximales pour les grandes images de 2005 et de 2003 sont 239 et 255 respectivement. Les ratios de taille entre les grandes images et les petites images de 2005 et de 2003 sont 3 et 4 respectivement. La Figure 4.23 montre que la moyenne des taux d'erreur des pixels visibles calculés avec les cas N° 1, N° 2, N° 3 et N° 4 définis plus tôt.

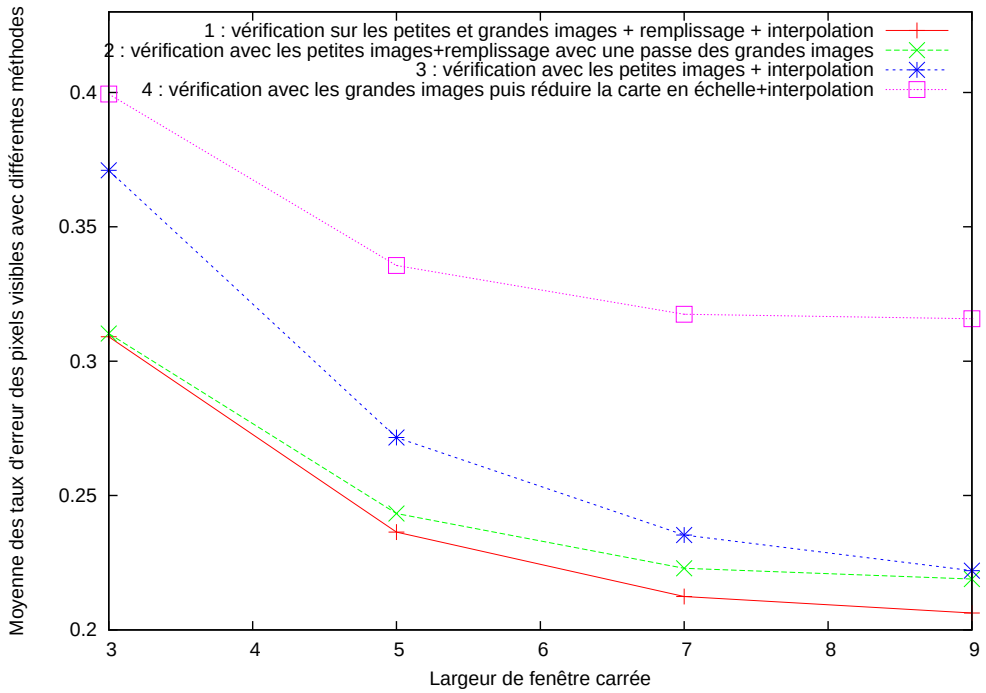


Figure 4.23 Comparaisons des moyennes des taux d'erreur des pixels visibles avec des combinaisons de techniques pour 8 paires d'images

Cette méthode d'interpolation est rapide. La partie de temps de l'interpolation MoyenPlusPropagationDuFond pour la création de la carte du cas N° 1 dans la Figure 4.20 prend 2,3 *ms* sur un processeur Intel® Core™ i7-2600K @ 3,40 GHz sous Windows 7 Professional. Nous n'avons pas fait l'implémentation en GPGPU.

Nous pouvons également remarquer que le remplissage de disparités des images en plus haute définition après la vérification de consistance a le meilleur résultat. Pour raison de performance, nous employons le remplissage de disparités des images en plus haute définition d'une seule passe dans la plupart de cas.

4.10 Autres aspects considérés

4.10.1 Méthodes de différence quadratique et de différence absolue

Nous calculons la valeur minimale de coût pour trouver la disparité. Le calcul de coût peut être fait par la différence quadratique ou par la différence absolue. La méthode avec la différence quadratique est plus précise que la méthode avec la différence absolue. Nous avons employé la différence quadratique puisque nous pouvons profiter de la capacité de calcul du GPGPU. La Figure 4.24 montre des comparaisons entre les taux d'erreur calculés par la

somme des différences au carré (SDC) ou la somme des différences absolues (SDA).

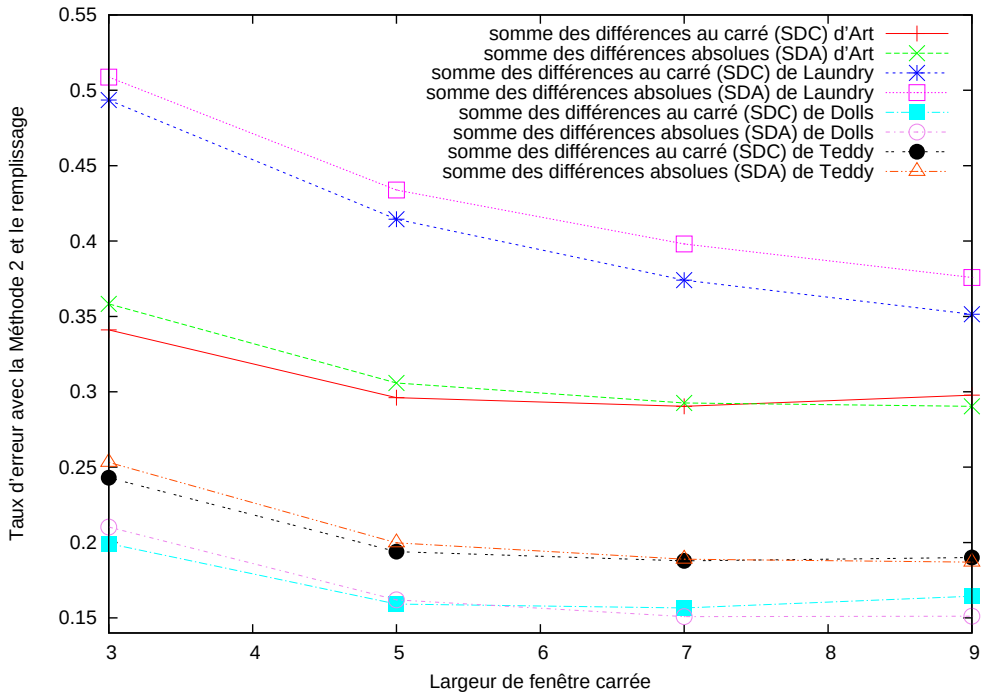


Figure 4.24 Comparaisons des taux d'erreur des pixels visibles avec la Méthode 2 et le remplissage pour 8 paires d'images

Toutefois, Hirschmüller et Scharstein ont indiqué que les désavantages de différence absolue peuvent facilement être compensés par la vérification de consistance et le remplissage ou de fortes contraintes de lissage (voir Hirschmüller et Scharstein, 2009, p. 6).

4.10.2 Méthodes de calcul de disparité de droite à gauche

Pour faire une vérification de consistance, on a besoin d'une passe de gauche à droite et d'une autre passe de droite à gauche. Quand on calcule la disparité de droite à gauche, on peut retourner les images ou simplement prendre les images telles quelles, puis échanger l'image de référence. Ces deux méthodes peuvent produire le même résultat. La méthode qui consiste à retourner les images est plus facile à utiliser parce qu'on peut employer le même programme. Nous voulons moins d'échange entre la mémoire globale et l'hôte, nous avons choisi la méthode de prendre les images telles quelles avec un seul transfert d'images de l'hôte vers la mémoire globale.

4.10.3 Discussion sur le remplissage

Nous voulons savoir quelle est la meilleure option entre le remplissage de disparités des images en haute définition et celui de disparités des mêmes images sous différentes conditions.

Dans la Figure 4.25, nous avons comparé les taux d'erreur des pixels visibles de la carte de disparité selon que nous la remplissions avec une carte de disparité créée avec de petites images ou que nous la remplissions avec une carte de disparité créée avec des images en plus haute définition. Pour cela, nous utilisons une taille de fenêtre T_f de 2 de plus pour les mêmes images et 3 fois plus grande pour les grandes images. Nous pouvons voir que le remplissage avec les images en plus haute définition peut améliorer la précision. Dans nos études, si la taille de fenêtre pour les images en plus haute définition est augmentée, le taux d'erreur diminue, mais cette tendance a sa limite. Une trop grande taille de fenêtre va augmenter le taux d'erreur, et une grande taille de fenêtre entraîne une plus longue durée d'exécution. Il ne faut donc pas trop l'augmenter.

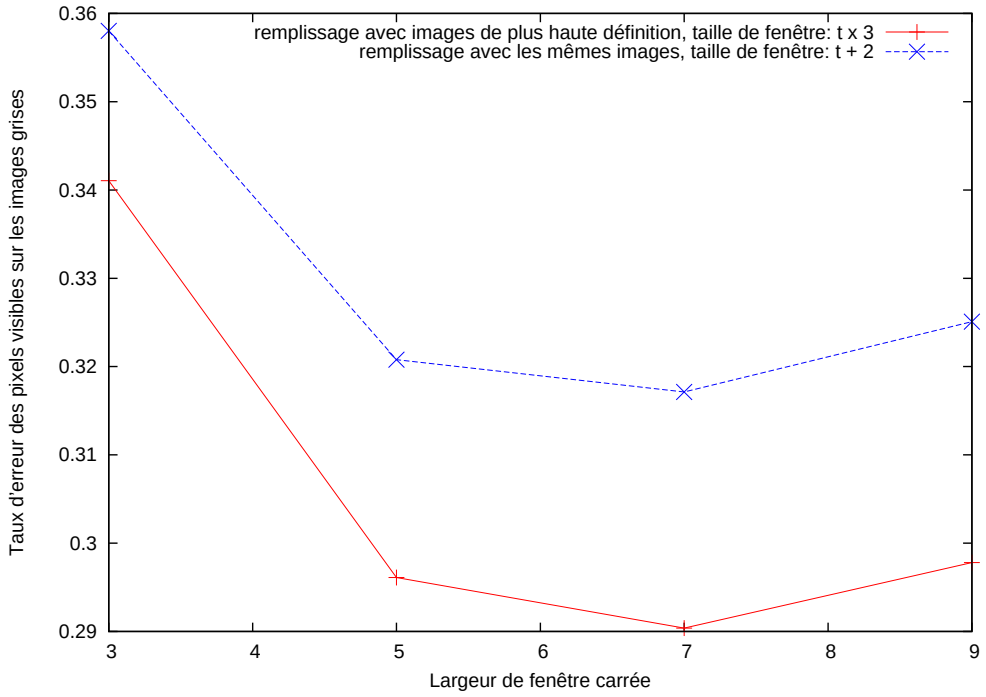


Figure 4.25 Taux d'erreur des pixels visibles de remplissage avec des images de différentes définitions de « *Art* »

Pour les images de haute définition, il se peut que certains détails détériorent la performance. Dans ce dernier cas, nous pouvons employer un filtre bilatéral pour neutraliser les petits changements.

4.10.4 Images en couleur et images grises

Hirschmüller et Scharstein ont indiqué que l'emploi des couleurs des images n'améliore presque pas la précision. Le coût basé sur l'intensité de contraste des images grises est plus constant que celui avec les images en couleur, car celles-ci présentent des variations de couleur causées par le prétraitement des appareils photo (voir Hirschmüller et Scharstein, 2009, p. 14).

Pour confirmer et aussi pour évaluer la différence entre deux modes d'images, nous avons fait quelques tests pour étudier l'impact de la différence entre les images en couleur format RVB et les images grises. Les images traitées sont les petites images de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les petites images de *Teddy* de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003), les tailles sont 463×370 et 450×375 respectivement.

Dans la Figure 4.26, nous avons comparé les taux d'erreur des pixels visibles après la vérification de consistance plus l'interpolation MoyenPlusPropagationDuFond des images grises par rapport aux images en couleur. Nous pouvons voir que les images en couleur traitées selon trois couches RVB auront une meilleure précision sur les images de « *Art* ». Par contre, nous avons aussi observé qu'une détérioration de précision (ou une augmentation de taux d'erreur) sur certaines images en couleur, par exemple, les images de *Teddy*. La détérioration peut être due aux particularités des images.

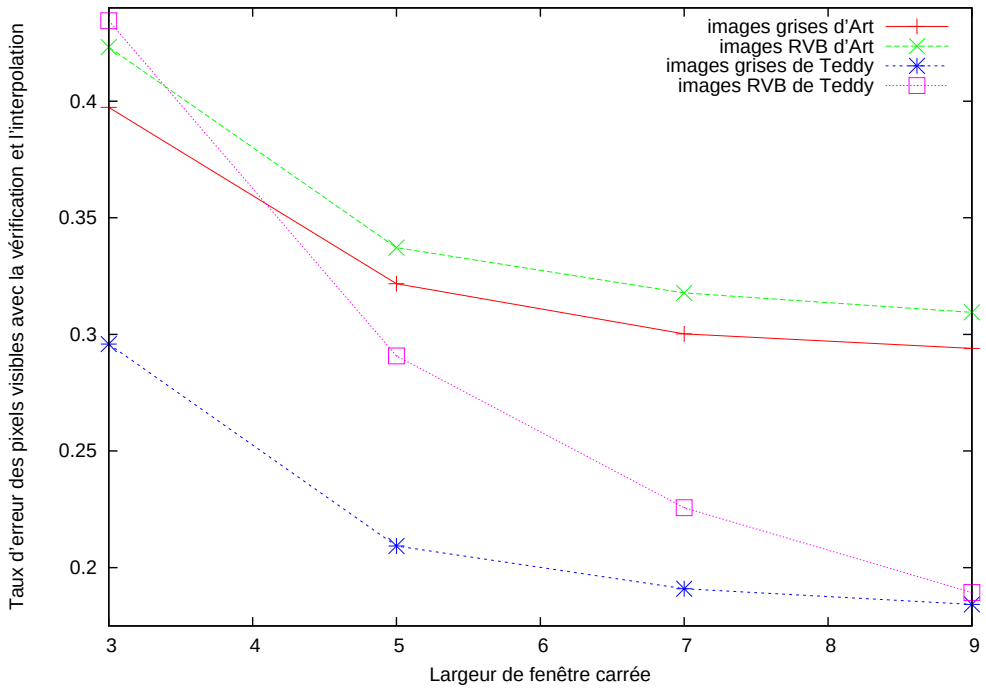


Figure 4.26 Taux d'erreur des pixels visibles en deux modes en fonction de la taille de la fenêtre avec les images de « *Art* »

Nous pouvons aussi observer que le taux d'erreur diminue jusqu'à un certain seuil. Une fois passé ce seuil, l'augmentation de taille de la fenêtre n'améliore pas la précision. Cela paraît évident parce qu'une trop grande fenêtre regroupe plus de bruit dans une comparaison de correspondance.

Le traitement en trois couches RVB a un coût. Dans la Figure 4.27, nous avons montré le changement de taux d'accélération selon la taille de la fenêtre en deux modes sources sur les petites images de « *Art* ». Nous pouvons constater que les images en couleur RVB peuvent ralentir le traitement d'à-peu-près de moitié. Le coût est élevé, mais nous pouvons aussi constater qu'avec GPGPU, le traitement en trois couches RVB a quand même eu une bonne accélération. Il faut noter que le calcul des écarts quadratiques de l'implémentation en CPU est réduit au minimum afin de mieux rendre compte de l'effet d'accélération. Il faut noter que la comparaison de taux d'accélération se fait des images grises GPGPU contre CPU et des images en couleur GPGPU contre CPU, donc il n'y a pas de ratio valide entre les deux courbes.

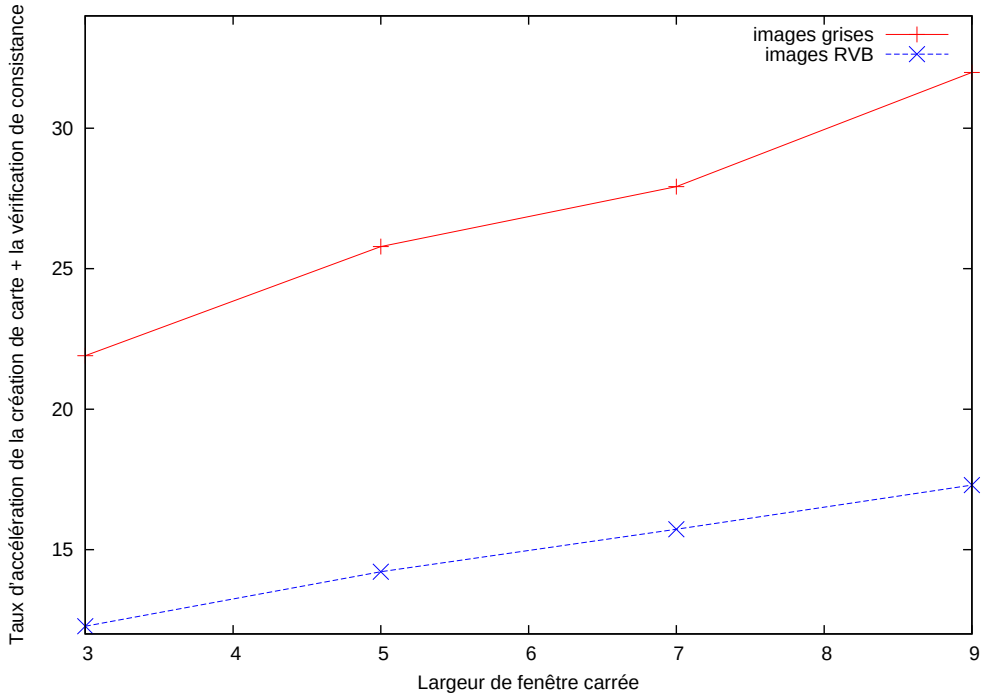


Figure 4.27 Taux d'accélération en deux modes (en RVB ou en nuances de gris) en fonction de la taille de la fenêtre sur les images de 463×370

Bien que souvent le mode en couleur soit meilleur en terme de précision, pour une meilleure performance, nous employons dans la plupart des cas les images grises.

4.10.5 Discussion sur le format interne d'image

Quand nous avons converti les images RVB en images grises, nous avons eu deux choix : les convertir en format "unsigned char" ou en format "float". Le format "float" occupe 4 octets pour chaque élément. Le format "float" est plus précis, mais demande plus de mémoire. Le format "float" en GPGPU peut demande un petit peu plus de temps par rapport au format "unsigned char", tandis que la différence du temps d'exécution sur CPU peut varier au moins deux fois plus entre le format "float" et le format "unsigned char". Nos études nous permettent de constater que le format "float" n'apporte ni d'augmentation évidente de précision de carte de disparité, ni d'amélioration par rapport au coût tant sur la mémoire que sur le temps d'exécution.

4.10.6 Comparaisons de performance de la Méthode 1 avec une implémentation d'OpenCV

Nous allons comparer la performance de la Méthode 1 et l'implémentation de la mise en correspondance par bloc en GPGPU d'OpenCV. Les images traitées sont les petites images de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005), la taille est 463×370.

La disparité maximale est 63 dans ces études, ce qui la situe dans une gamme de [0, 63]. Pour une fenêtre de comparaison de taille 19, le temps d'OpenCV est en moyenne à peu près égale à la moitié du temps avec la Méthode 1 de notre implémentation. Nous supposons que cela est dû à la différence des détails d'implémentation, au fait que notre implémentation se base sur OpenCL et que celle d'OpenCV se base sur CUDA C. Une plus grande fréquence d'accès à la mémoire globale combinée avec le nombre de fils d'exécution peut souvent expliquer l'augmentation de temps d'exécution. Quant à la différence de plateforme, OpenCL est normalement un petit peu plus lent que CUDA C parce qu'il y a un peu plus de couches de traduction.

Tableau 4.5 Comparaisons des performances avec OpenCV

Taille de fenêtre	Programme concerné	Temps (ms)
3	OpenCV	1,3
	Méthode 1	2,0
19	OpenCV	1,7
	Méthode 1	4,4

CHAPITRE 5

AMÉLIORATION DE LA QUALITÉ

Nous allons continuer à essayer sur plus d'images, avec d'autres méthodes ou de combinaisons de méthodes pour améliorer la qualité d'une carte de disparité. Comme ce sont des essais, nous n'avons pas tout implémenté en GPGPU.

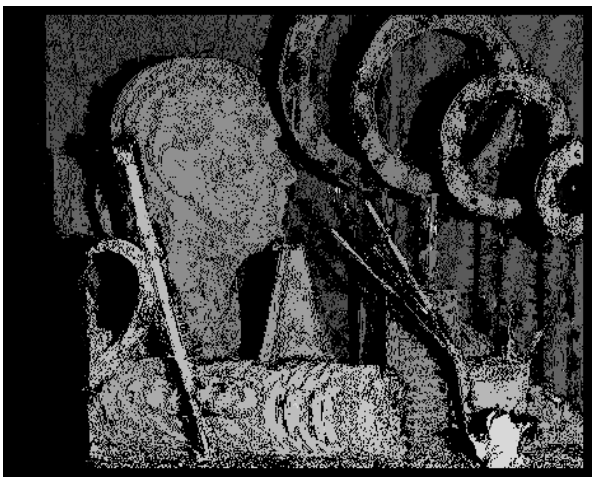
5.1 Études avec Census

Selon Hirschmüller et Scharstein, la méthode Census est la meilleure et la plus fiable dans l'ensemble concernant la qualité de correspondance des images ayant des différences radiométriques (voir Hirschmüller et Scharstein, 2009). Weber *et al.* ont présenté un travail dans ce domaine (voir Weber *et al.*, 2009, p. 790) que nous avons mentionné dans la sous-section 3.11.3.

5.1.1 Quelques cartes de disparité obtenues

Nous avons réalisé une implémentation en CPU et montré les résultats obtenus avec une fenêtre de taille 15 pour les petites images de taille 463×370 sous un seuil de vérification de consistance de 0 dans la Figure 5.1. La définition de la taille de la fenêtre T_f est comme celle de la mise en correspondance par bloc dans la section 3.9, soit la largeur d'un carré centré sur chaque pixel étudié. La disparité maximale est 79, 239 pour les petites images et les images en plus haute définition respectivement. Afin de faire la comparaison avec les expériences précédentes, nous employons 79 et 239. La méthode de remplissage de disparités des images en plus haute définition est semblable à celle introduite dans la sous-section 4.8.2, à cette différence près que la taille de la fenêtre T_f reste la même tant pour les petites images que pour les grandes. Pourtant, une trop grande ou trop petite fenêtre n'aidera pas à la précision. En plus, une trop grande fenêtre peut ralentir de beaucoup l'exécution. Les bordures noires autour des cartes de disparité sont invalidées parce que nous n'avons pas de valeur selon notre méthode de comparaison de coût de la Census.

La Figure 5.2 montre les résultats après avoir passé l'interpolation MoyenPlusPropagationDuFond sur les cartes de disparité introduites dans la Figure 5.1. Visiblement, pour le cas avec remplissage, on ne voit pas la différence parce qu'avec une seule passe sur les grandes images, les trous existants sont déjà remplis normalement.



vérification de consistance sur les petites

vérification de consistance sur les petites + remplissage de disparités de HD après une seule passe sur les images HD

Figure 5.1 Comparaison des cartes obtenues sans interpolation

vérification de consistance avec les petites + interpolation

vérification de consistance sur les petites + remplissage après une seule passe sur les grandes images + interpolation

Figure 5.2 Comparaison des cartes obtenues avec Census et interpolation

5.1.2 Comparaisons de combinaisons de méthodes basées sur Census

Nous avons présenté les taux d'erreur des pixels visibles et les taux d'erreur globaux dans le Tableau 5.1. Les cartes de disparité ont été augmentées à l'échelle pour un meilleur affichage. Nous pouvons voir que la précision peut augmenter avec la méthode de remplissage de disparités des images en plus haute définition. Les tailles de la fenêtre pour les petites et

grandes images sont identiques, toutes de 15. Le terme « remplissage HD » signifie le remplissage de disparités des images en plus haute définition après une vérification de consistance. « Interpolation » signifie l'interpolation MoyenPlusPropagationDuFond introduite dans la sous-section 4.8.5. Le symbole « + » signifie la combinaison des deux options.

Tableau 5.1 Comparaisons des performances avec Census

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur globaux
Vérification de consistance sur les cartes de disparité avec les petites images + interpolation	20,53%	35,42%
Vérification de consistance sur les cartes de disparité avec les petites images + remplissage avec des disparités d'une seule passe des images en plus haute définition avec les tailles de fenêtre identiques de 15	16,46%	35,72%
Vérification de consistance sur les cartes de disparité avec toutes les tailles d'images + remplissage HD avec les taille de fenêtre identiques de 15 + interpolation	16,29%	31,51%

Nous pouvons observer une très bonne précision si nous employons la méthode de remplissage de disparités des images en plus haute définition combinée avec l'interpolation MoyenPlusPropagationDuFond. Les taux d'erreur sont fournis comme référence. À cause des particularités de la méthode Census, les bordures d'une hauteur de $r = \text{plancher}(T_f)$ en haut et en bas, d'une largeur de r à gauche et à droite autour de la carte sont exclues du calcul des taux d'erreur. Ces parties sont incluses dans le calcul de la Méthode 0 et Méthode 2 présenté plus haut.

Afin de réduire le temps d'exécution, nous pouvons n'exécuter dans le remplissage qu'une partie de la création de carte de disparité en employant des images en plus haute définition, c'est-à-dire nous ne chercherons que les pixels correspondants aux ceux invalides après la vérification de consistance sur les petites cartes de disparité.

5.2 Combinaison de remplissage et d'interpolation

Nous allons présenter quelques résultats en employant la combinaison de remplissage de disparités des images en plus haute définition et d'interpolation sur différentes images pour montrer ce que ces méthodes peuvent donner.

5.2.1 Combinaison basée sur la Méthode 2

Nous avons aussi intérêt à connaître les taux d'erreur des différentes images avec la combinaison de la Méthode 2, de la vérification de consistance sur les petites et grandes images, du remplissage et de l'interpolation MoyenPlusPropagationDuFond. La Figure 5.3 montre des comparaisons avec les images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Les images de 2005 employées incluent les plus grandes et les plus petites de « *Art* », de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». Pour ce qui est des images de 2003, elles incluent les images les plus grandes et les plus petites de « *Cones* » et de « *Teddy* ». La disparité maximale des petites images de 2005 est 79, celle des petites images de 2003 est 63. Les disparités maximales pour les grandes images de 2005 et de 2003 sont 239 et 255 respectivement. Les ratios de taille entre les grandes images et les petites images de 2005 et de 2003 sont 3 et 4 respectivement.

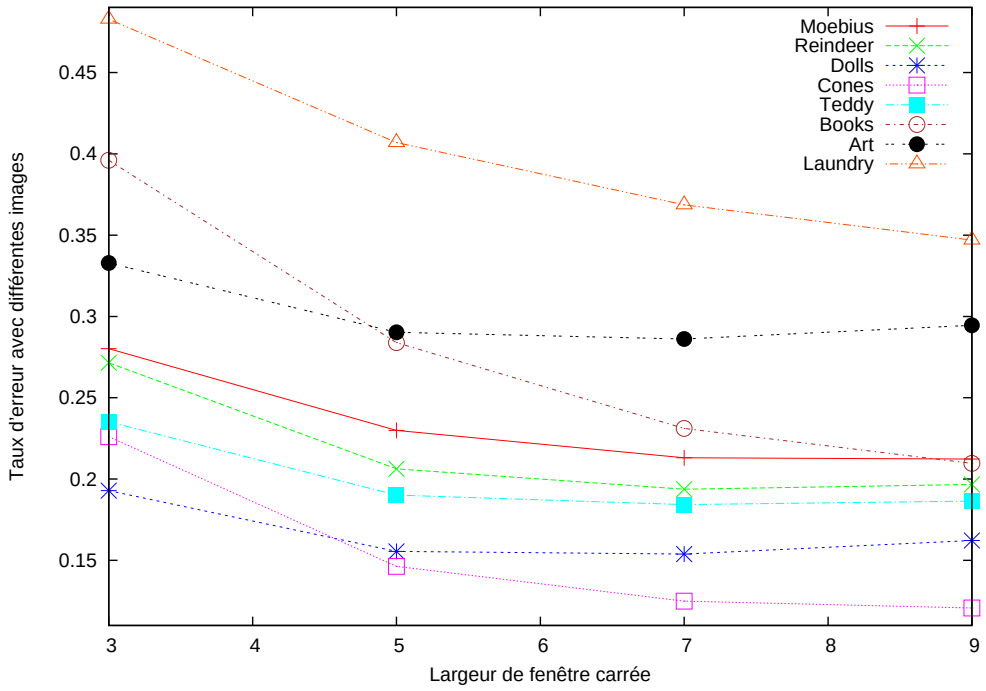


Figure 5.3 Comparaisons des taux d'erreur des pixels visibles avec différentes images et une combinaison d'options basées sur la Méthode 2

Nous pouvons remarquer une grande variation de taux d'erreur selon les images employées. Nous considérons qu'une analyse des structures d'images sera une voie possible de régler le problème.

5.2.2 Combinaison basée sur la méthode Census

La Figure 5.4 montre les comparaisons avec les images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003) en employant la méthode Census, la vérification de consistance sur les petites images, le remplissage de disparités d'une passe avec les grandes images et l'interpolation MoyenPlusPropagationDuFond. Comme le cas de la Figure 5.3, les images de 2005 employées incluent les plus grandes et les plus petites de « *Art* », de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». Pour ce qui est des images de 2003, elles incluent les images les plus grandes et les plus petites de « *Cones* » et de « *Teddy* ». La disparité maximale des petites images de 2005 est 79, celle des petites images de 2003 est 63. Les disparités maximales pour les grandes images de 2005 et de 2003 sont 239 et 255 respectivement. Les ratios de taille entre les grandes images et les petites images de 2005 et de 2003 sont 3 et 4 respectivement. La taille de fenêtre T_f reste en proportion avec la taille des images, c'est-à-dire que T_f pour les grandes images sera $T_f \times r$.

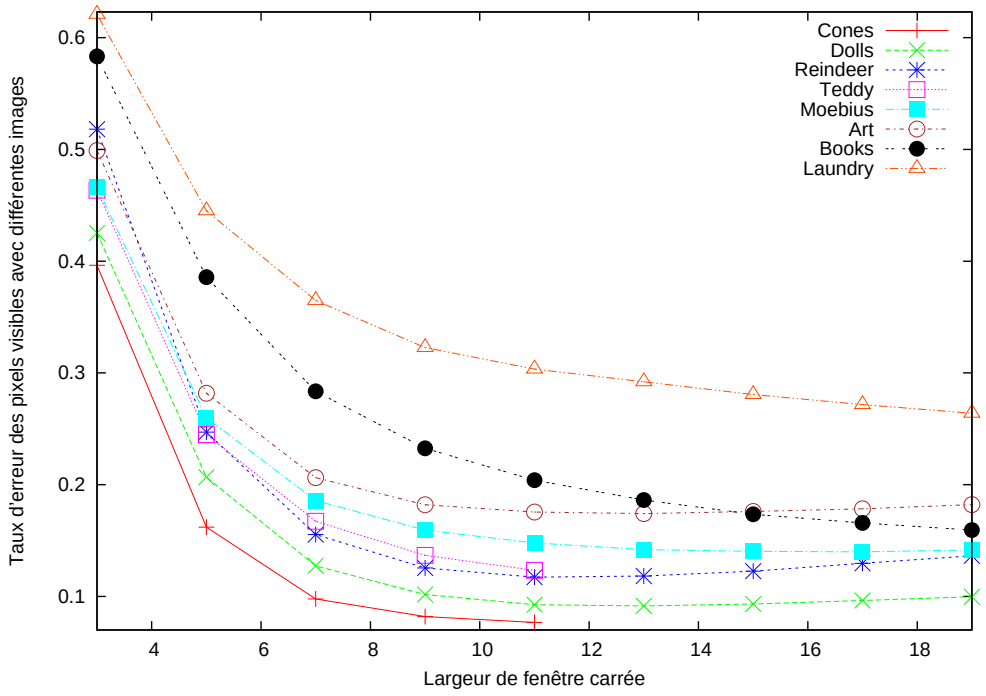
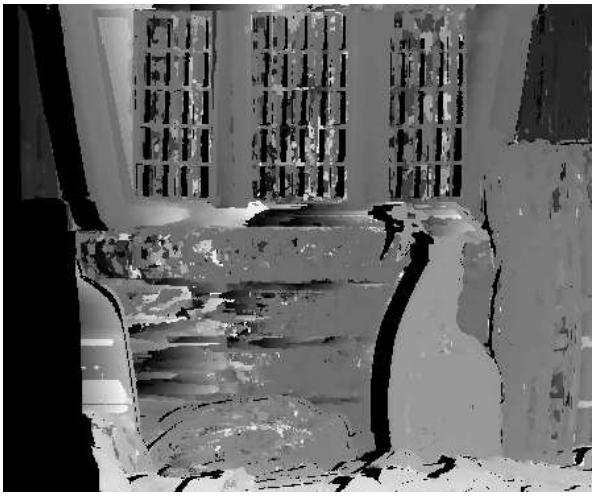


Figure 5.4 Comparaisons des taux d'erreur avec différentes images et une combinaison d'options basées sur la méthode Census

Nous pouvons remarquer que la combinaison basée sur la méthode Census a une meilleure qualité que celle basée sur la Méthode 2, toutefois, les images de « *Laundry* » restent parmi les plus difficiles à traiter.

5.2.3 Analyse des résultats avec les images de *Laundry*

La Figure 5.5 montre les cartes obtenues en employant les combinaisons de remplissage et d'interpolation basées sur la Méthode 2 et la méthode Census avec les images de « *Laundry* ». Bien qu'avec la taille 7, il y ait moins d'erreurs qu'avec la taille 5 dans le cas de la Méthode 2, nous avons montré la carte obtenue à partir de la taille 5 parce qu'elle est moins étirée horizontalement par l'effet de la mise en correspondance par bloc que nous avons introduit dans la sous-section 3.5.2. La taille de fenêtre pour le cas de Census est 15. La taille de fenêtre pour les grandes images augmente en proportion avec le ratio r entre la taille des grandes images et celle des petites images.



basée sur la Méthode 2, taille : 5,
taux d'erreur des pixels visibles : 40,69%

basée sur Census, taille : 15,
taux d'erreur des pixels visibles : 28,07%

Figure 5.5 Comparaison des résultats de *Laundry* avec combinaisons d'options basées sur la Méthode 2 et la méthode Census

Afin de comprendre ce qui complique le traitement des images de « *Laundry* », nous avons affiché l'image originale et la carte de disparité de référence dans la Figure 4.1.

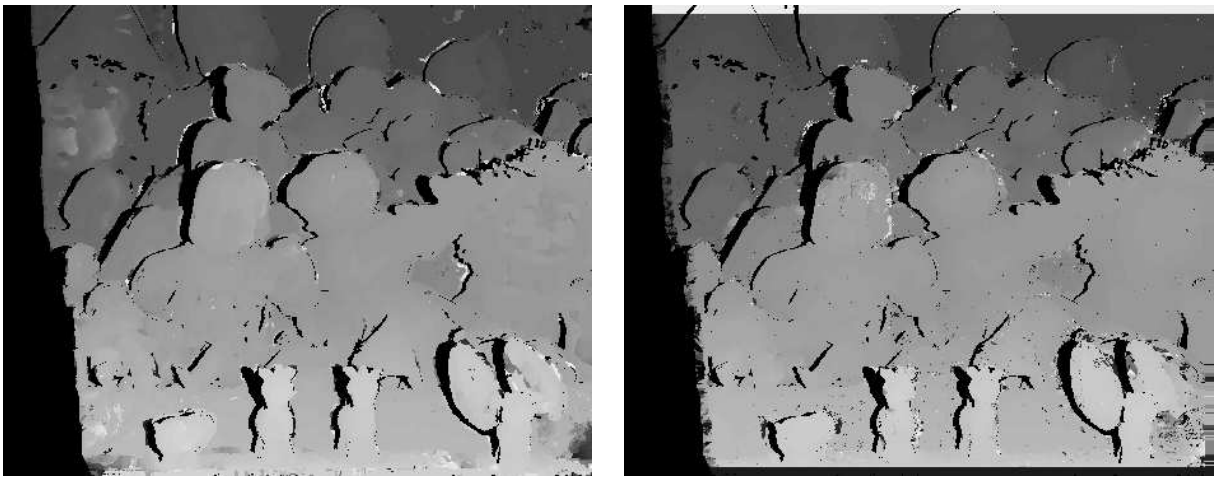
Nous avons constaté qu'il est difficile de traiter les grilles répétitives et détaillées avec la mise en correspondance par bloc ; la qualité est moins bonne que celle obtenue avec la méthode Census. Les lignes verticales sur le côté d'un panier à linge, les couleurs changeantes des objets dans le panier, la texture de bois sous le panier et la couleur presque inchangée du conteneur de détergent ont tout rendu les comparaisons des sommes des différences quadratiques difficiles. À cause de la petite taille des détails, la méthode Census présente aussi de difficultés.

Nous avons employé les mêmes paramètres partout durant les tests. Pour améliorer la qualité, nous pouvons choisir un autre groupe de paramètres, employer différentes options et ajouter des traitements supplémentaires, toutefois, nous pensons que ces mesures ont des

limites. Nous supposons qu'il faut des mesures plus avancées, incluant analyse des formes et segmentation, etc.

5.2.4 Analyse des résultats avec les images de *Dolls*

La Figure 5.6 montre les cartes obtenues en employant les combinaisons basées sur la Méthode 2 et la méthode Census avec les images de « *Dolls* ». Visuellemnt la combinaison basée sur la Méthode 2 produise une carte plus lisse par manque de petits points foncés. Nous pouvons lisser la carte obtenue à base de la méthode Census en enlevant des petites zones isolées, par exemple avec l'option d'enlèvement de taches introduite dans la sous-section 4.8.3. Les tailles de fenêtre sont 5, 15 pour la Méthode 2 et Census respectivement. La taille de fenêtre pour les grandes images augmente en proportion avec le ratio r entre la taille des grandes images et celle des petites images.



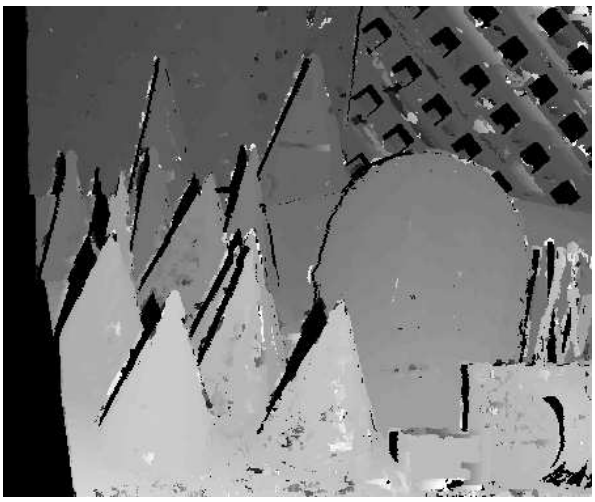
basée sur la Méthode 2, taille : 5,
taux d'erreur des pixels visibles : 15,55%

basée sur Census, taille : 15,
taux d'erreur des pixels visibles : 9,32%

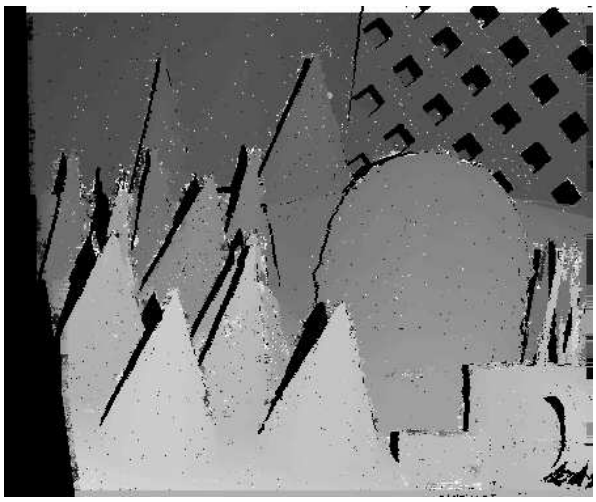
Figure 5.6 Comparaison des résultats de *Dolls* avec combinaisons d'options basées sur la Méthode 2 et la méthode Census

5.2.5 Analyse des résultats avec les images de *Cones*

La Figure 5.7 montre les cartes obtenues en employant les combinaisons basées sur la Méthode 2 et la méthode Census avec les images de « *Cones* ». Les plus grandes images ont une taille 4 fois plus grande que celle des petites images. Les tailles de fenêtre sont 5, 11 pour la Méthode 2 et Census respectivement. La taille de fenêtre pour les grandes images augmente en proportion avec le ratio r entre la taille des grandes images et celle des petites images.



basée sur la Méthode 2, taille : 5,
taux d'erreur des pixels visibles : 14,63%



basée sur Census, taille : 11,
taux d'erreur des pixels visibles : 7,67%

Figure 5.7 Comparaison des résultats de *Cones* avec combinaisons d'options basées sur la Méthode 2 et la méthode Census

Comme les analyses de la sous-section 5.2.3, les grands carreaux répétitifs en forme de losange ont rendu le traitement difficile, toutefois, le résultat est meilleur tant visuellement que numériquement qu'avec les images de « *Laundry* » si en référant les chiffres de la Figure 5.4 et les cartes de disparité de la Figure 5.5 parce que les carreaux de la grille ne sont ni horizontaux ni verticaux et le fond derrière les carreaux ne varie pas beaucoup.

5.2.6 Choix de la taille de fenêtre pour Census en employant les grandes images

La Figure 5.8 montre les comparaisons avec les images comme celles employées dans la Figure 5.4. La méthode de remplissage de disparités des images en plus haute définition est semblable à celle employée dans la Figure 5.4, à cette différence près que la taille de la fenêtre T_f reste la même tant pour les petites images que pour les grandes. Tandis que dans la Figure 5.4 la taille de fenêtre T_f reste en proportion avec la taille des images, c'est-à-dire que T_f pour les grandes images sera $T_f \times r$. C'est la particularité de la méthode Census : une trop grande ou trop petite fenêtre n'aidera pas à la précision. En plus, une trop grande fenêtre peut ralentir de beaucoup l'exécution. Nous pouvons voir que le taux d'erreur peut diminuer encore pour une grande fenêtre en comparaison avec ceux de la Figure 5.4.

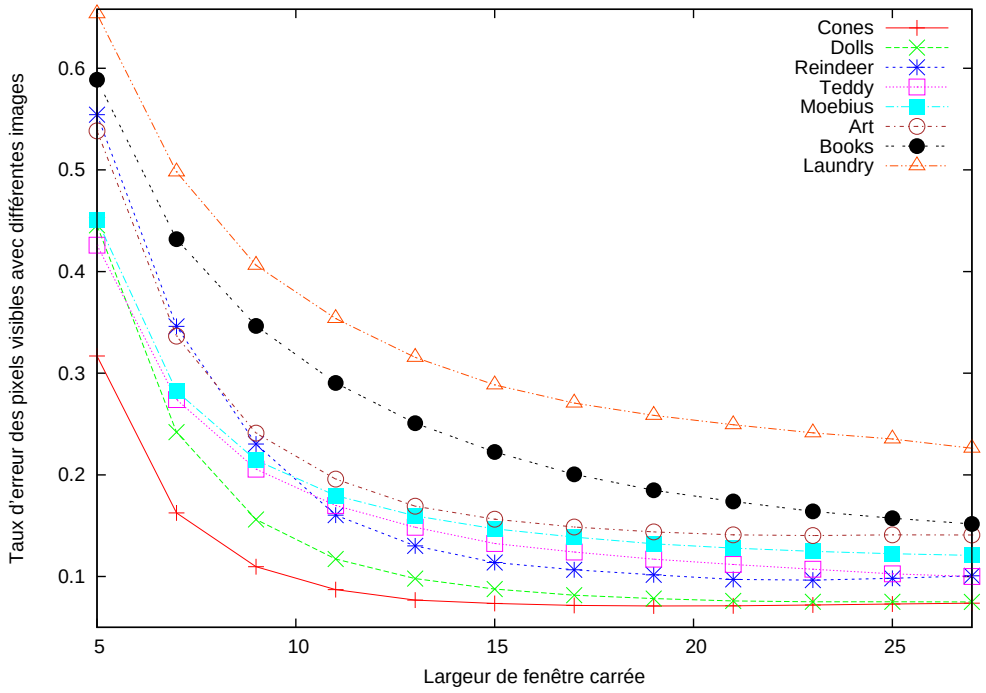


Figure 5.8 Comparaisons des taux d'erreur avec différentes images et une combinaison d'options basée sur la méthode Census avec la même taille de fenêtre pour les petites et grandes images

Nous avons souvent employé les mêmes tailles de fenêtre tant sur les petites images que sur les grandes images pour la méthode Census, et nous avons employé principalement la taille de fenêtre $T_f = 15$ dans nos études.

5.3 Amélioration de la qualité avec les images en plus haute définition

Nous avons intérêt à connaître l'amélioration de la qualité en employant les images en plus haute définition. Nous avons recherché cette possible amélioration en employant les images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003) et en procédant à la vérification de consistance sur les petites, et le remplissage après une passe des grandes images combinée à l'interpolation MoyenPlusPropagationDuFond, tout cela basé respectivement sur la Méthode 2 et sur la méthode Census. Les images de 2003 incluent la série d'images de « *Cones* » et de « *Teddy* » : les plus petites, les 2 fois plus grandes et les 4 fois plus grandes. Les disparités maximales des petites images, des 2 fois plus grandes images et des 4 fois plus grandes images sont respectivement de 63, 127 et 255.

5.3.1 Amélioration de la qualité avec combinaisons basées sur la Méthode 2

La Figure 5.9 montre l'amélioration de la qualité avec les images en plus haute définition en employant une combinaison basée sur la Méthode 2. L'emploi des images 4 fois plus grandes donne presque toujours un meilleur résultat.

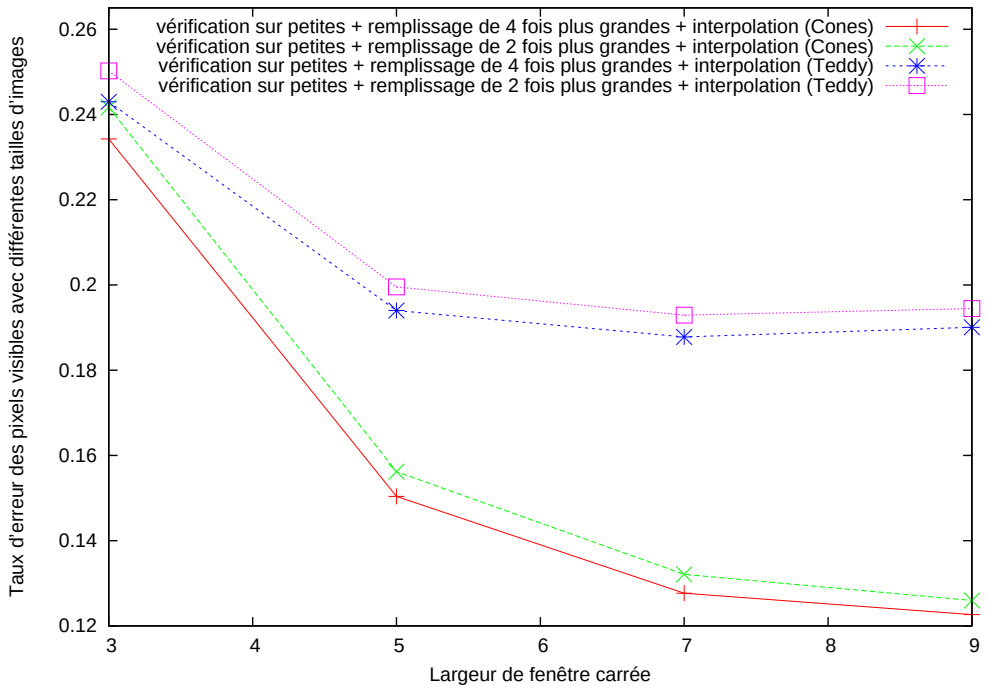


Figure 5.9 Comparaisons des taux d'erreur avec différentes tailles d'images et une combinaison d'options basées sur la Méthode 2

5.3.2 Amélioration de la qualité avec combinaisons basées sur la méthode Census

La Figure 5.10 montre l'amélioration de la qualité avec les images en plus haute définition en employant la combinaison basée sur la méthode Census. Cette combinaison procède la vérification de consistance avec les petites images puis une passe de gauche à droite avec les grandes images, et dernièrement l'interpolation MoyenPlusPropagationDuFond. Nous pouvons également observer que l'emploi des images 4 fois plus grandes donne presque toujours un meilleur résultat. Cela prouve partiellement que les images en plus haute définition peuvent contribuer à augmenter la précision de la carte de disparité.

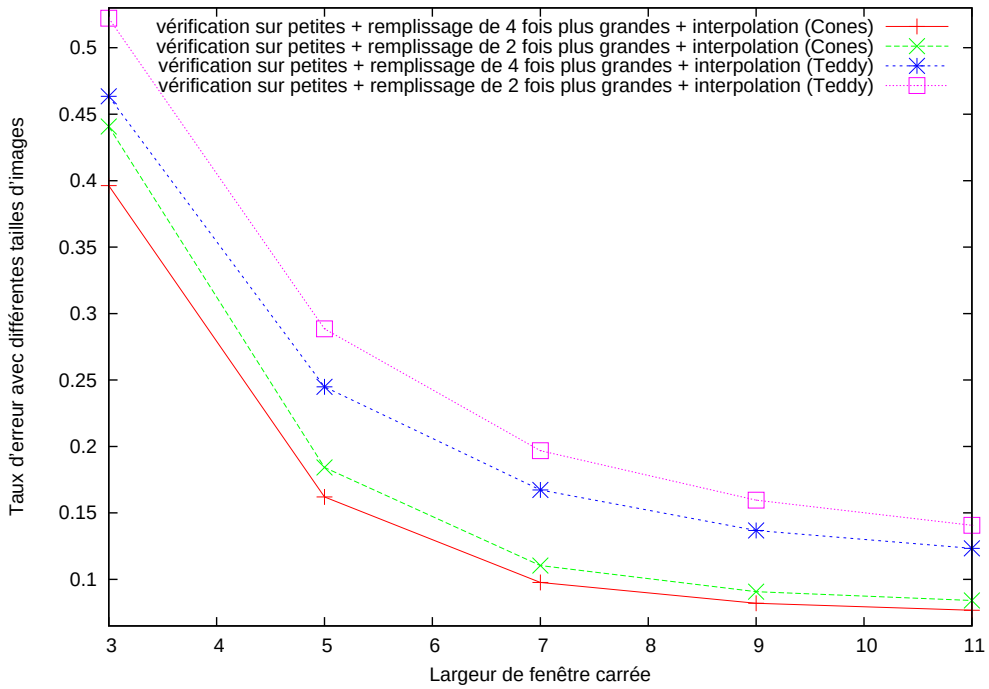


Figure 5.10 Comparaisons des taux d'erreur avec différentes tailles d'images et une combinaison d'options basées sur la méthode Census

5.3.3 Potentiel d'amélioration de la qualité avec d'images en plus haute définition

La vérification de consistance invalide beaucoup de pixels. Nous avons évalué le potentiel d'amélioration avec les images en plus haute définition en employant les images de Middlebury de 2005 présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005) et les plus petites et les 4 fois plus grandes images de Middlebury de 2003 présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003). Nous employons le remplissage de disparités des images en plus haute définition après la vérification de consistance pour combler les pixels invalides. Nous avons employé la taille de fenêtre de 3 et $3r$ sur les petites images et les grandes images respectivement pour la Méthode 2, la taille de 15 sur toutes les images pour Census. Si nous considérons que les disparités après la vérification de consistance est fiables et que le remplissage consiste d'une correction, pour la Méthode 2 et Census respectivement, en moyenne, 39,85% et 41,51% de pixels sont potentiels d'être corrigés, la correction réelle est 41,22% et 27,09% sur les pixels potentiels, ou 16,08% et 11,07% sur tous les pixels visibles.

Cela prouve que nous pouvons faire un calcul sélectif sur les images en plus haute définition, nous permet de réduire des erreurs tout en évitant le calcul coûteux sur les images stéréoscopiques tout entières.

5.4 Combinaison de Census et de mise en correspondance par bloc

Nous avons employé une combinaison de Census et de mise en correspondance par bloc avec la somme des différences au carré (SDC) pour profiter des avantages des deux approches de calcul de coût. Nous avons employé une taille de fenêtre fixe de 3 pour les petites images et 9 ou 12 selon le ratio de tailles des images pour les grandes images concernant la partie de la mise en correspondance par bloc dans nos études pour simplifier la comparaison. Le choix de la taille de 3 est arbitraire, nous pouvons choisir une meilleure taille en employant la méthode heuristique. La combinaison se fait simplement avec l'équation 5.1 et $\beta = 0,2$ dans le but d'une simplification. La définition des paramètres β dépend de l'application ; le principe est de ramener deux coûts globalement au même niveau afin de ne pas influencer le coût combiné par un coût en particulier.

$$C = C^2_{\text{Census}} + \beta \cdot C_{\text{SDC}} \text{ où } C_{\text{Census}} \text{ est le coût de Census et } C_{\text{SDC}} \text{ est le coût de BM, } \beta \text{ est un paramètre. Dans nos études, } \beta = 0,6 \text{ pour les petites images, } \beta = 0,2 \text{ pour les grandes images} \tag{5.1}$$

Mei *et al.* ont employé l'équation 5.2 comme la fonction de combinaison. (voir Mei *et al.*, 2011, p. 2) λ est un paramètre.

$$\rho(c, \lambda) = 1 - \exp(-\frac{c}{\lambda}) \tag{5.2}$$

La réussite avec l'équation 5.2 dépend totalement au choix du paramètre λ qui dépend lui-même de l'application. Notre fonction 5.1 est normalement un peu plus générale. De plus, l'équation 5.2 normalement prend deux fois plus de temps que l'équation 5.1 parce que l'équation 5.2 calcule une division puis une exponentielle.

Quand nous avons appliqué la méthode de remplissage, le résultat est moins bon en comparaison avec celui de Census. Pour élucider cette situation, nous avons étudié le résultat de la vérification de consistance en employant les petites images dans le Tableau 5.2. Le seuil de double vérification est 0. βBM+Census^2 signifie d'une seule passe de gauche à droite de la combinaison de Census et de mise en correspondance par bloc (BM) en employant la fonction de combinaison $C = C^2_{\text{Census}} + \beta \cdot C_{\text{SDC}}$.

Tableau 5.2 Comparaisons des taux d'erreur de la vérification de consistance en employant la combinaison de Census et de mise en correspondance par bloc sur les petites images de « *Art* »

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur
SANS interpolation		
βBM+Census^2	53,57%	66,79%
Census	43,40%	58,48%
SANS interpolation et pour les pixels n'ayant pas de zéro		
βBM+Census^2	25,81%	29,84%
Census	8,44%	11,87%
AVEC interpolation		
βBM+Census^2	37,03%	48,38%
Census	20,53%	35,42%

Nous pouvons remarquer un taux d'erreur exceptionnellement élevé pour βBM+Census^2 sans interpolation et pour les pixels n'ayant pas de zéro. Afin d'éclairer la situation, nous avons listé dans le Tableau 5.3 quelques chiffres si nous faisons une seule exécution de gauche à droite. Nous pouvons remarquer que les taux d'erreur sont quand même dans presque le même niveau, de plus, le taux d'erreur de βBM+Census^2 est moins élevé que celui de Census, c'est bien contraire de ce que nous avons observé dans le Tableau 5.2.

Tableau 5.3 Comparaisons des taux d'erreur d'une seule exécution en employant la combinaison de Census et de mise en correspondance par bloc sur les petites images de « *Art* »

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur
SANS interpolation		
βBM+Census^2	18,46%	37,63%
Census	23,55%	41,59%
SANS interpolation et pour les pixels n'ayant pas de zéro		
βBM+Census^2	18,41%	34,86%
Census	24,78%	40,83%

Cette observation inhabituelle ne peut pas être généralisée sans des études plus profondes. Pourtant, Il ne sera pas recommandé d'employer la méthode de la vérification de consistance puis le remplissage dans le cas de la combinaison de Census et de mise en correspondance par bloc.

Nous pouvons voir plus tard dans le Tableau 5.7 et le Tableau 5.8 que cette combinaison de Census et de mise en correspondance par bloc peut diminuer le taux d'erreur des pixels

visibles dans le cas d'une seule passe, fonctionne bien tant pour les petites images que pour les grandes images.

5.5 Amélioration de la qualité à l'aide de MMC

Supposons que nous avons les symboles observés et les probabilités d'observation et de transition, notre problème est de connaître la séquence optimale d'états $D = d_1 d_2 \cdots d_n$ lorsque nous avons une séquence d'observations $O = o_1 o_2 \cdots o_n$. La D obtenue sera une ligne de la carte de disparité recherchée.

Si nous étudions la matrice de transition, nous pouvons normalement observer que cette matrice, dans la plupart de cas, ressemble plus à une matrice diagonale. Dans la plupart des cas nos observations nous permettent de constater que chaque pixel de disparité a tendance à conserver son état actuel. $P(d_i = i|d_i = i), 1 \leq i \leq N$ sera plus élevée que les autres probabilités en général. Si nous supposons encore que chaque autre état a la possibilité équivalente d'arriver à n'importe quel état, nous avons une matrice de transition créée avec un paramètre. La matrice la plus simple peut être :

$$\begin{cases} P(d_{i+1} = j|d_i = j) = 1 - \dfrac{N-1}{\sigma_t}, & 1 \leq i \leq N-1, 1 \leq j \leq N \\ P(d_{i+1} \neq j|d_i = j) = \dfrac{1}{\sigma_t}, & 1 \leq i \leq N-1, 1 \leq j \leq N \end{cases} \tag{5.3}$$

Dans ce dernier cas, la matrice de transition peut être définie par un seul paramètre σ. Nos expériences prouvent qu'elle peut s'adapter à plusieurs situations, contrairement à une matrice de transition obtenue à partir d'un seul type d'images à l'aide d'une méthode statistique.

Dans nos études, nous pouvons prendre le nombre d'observations égal à celui d'états. Donc, d'une façon similaire, nous pouvons définir une matrice d'émission comme :

$$\begin{cases} P(o_j = j|d_i = i) = 1 - \dfrac{M-1}{\sigma_e}, & i = j, 1 \leq i \leq N, 1 \leq j \leq M \\ P(o_j \neq j|d_i = i) = \dfrac{1}{\sigma_e}, & i = j, 1 \leq i \leq N, 1 \leq j \leq M \end{cases} \tag{5.4}$$

Afin de simplifier la présentation, nous avons pris $\sigma = \sigma_t = \sigma_t$ dans ce mémoire. Parce que ces paramètres ne se servent que pour le but d'évaluation.

Si nous n'avons pas de carte de référence, nous pouvons employer une matrice définie par un paramètre ou une matrice obtenue statistiquement à partir d'un autre groupe d'images similaires. Nous pouvons espérer avoir une meilleure qualité de carte de disparité lorsque nous employons une matrice bien définie par un certain paramètre au lieu d'une matrice créée pour

un autre groupe d'images.

Le plus important de cette méthode est de déterminer les matrices de transition et d'émission. Pour une certaine application, ces matrices suivent souvent de règles ou de caractéristique. Donc nous pouvons quand même employer cette méthode. Cette méthode peut se servir comme un préalable pour évaluer le potentiel de méthodes probabilistes.

5.5.1 Impact du seuil de vérification

Le Tableau 5.4 montre que les valeurs initiales doivent être précises. Pour cela, il faut utiliser un petit seuil de vérification de consistance qui permet d'éliminer les valeurs susceptibles d'être fausses. Dans nos études, comme il y a un nombre limité d'observations, leur influence sera élevée pour la qualité de MMC.

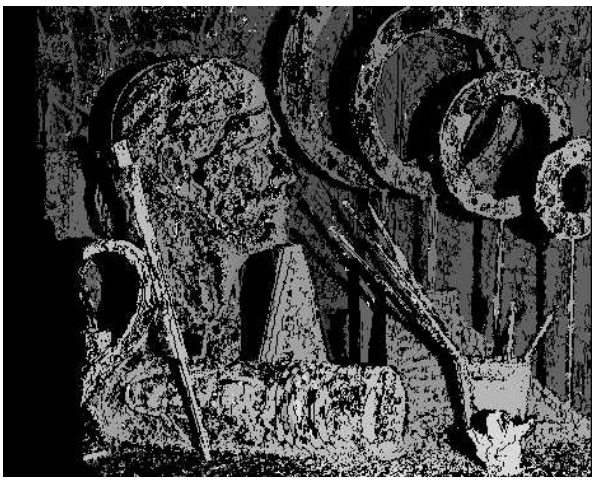
Nous avons employé la Méthode 2, avec la disparité dans la gamme [0, 79] et une taille de fenêtre 3. Les images employées sont les petites et grandes images de la taille 463×370 et 1390×1110 de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005). Les tests sont effectués sur NVIDIA® GeForce® GTX 580 et Intel® Core™ i7-2600K @ 3,40 GHz sous Windows 7 Professional. Intel® Core™ i7-2600K a 4 cœurs, mais l'exécution de référence en CPU s'exécute sur un seul cœur.

Tableau 5.4 Comparaisons des performances avec Viterbi

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur	G/CPU	Temps(s)	Accélération
Seuil de double vérification : 0					
Avec la vérification de consistance sur les petites images + interpolation	39,73%	49,11%	GPU	0,007	23,06
			CPU	0,156	
Après MMC	17,89%	24,01%	GPU	2,097	60,36
			CPU	126,563	
Avec la vérification de consistance sur les petites et grandes images + remplissage HD + interpolation	34,14%	44,43%	GPU	0,126	56,79
			CPU	7,128	
Après MMC	16,63%	23,86%	GPU	2,337	53,05
			CPU	124,000	
Seuil de double vérification : 1					
Avec la vérification de consistance + interpolation	39,17%	49,19%	GPU	0,007	23,85
			CPU	0,155	
Après MMC	20,75%	27,45%	GPU	2,218	55,86
			CPU	123,886	

5.5.2 Amélioration de la qualité avec le MMC

La Figure 5.11 montre le résultat d'exécution du MMC. Le seuil de vérification de consistance est 0. Les matrices de transition et d'émission sont calculées à partir de la carte de disparité de référence qui a une gamme de valeur [0, 255]. Nous n'avons pas fait le remplissage dans ce cas d'étude. Nous avons montré la carte après la vérification de consistance en employant les petites images à gauche. Évidemment, cette carte manque d'information. Le MMC peut très bien remplir les trous pour former une carte complète montrée à la droite de la Figure 5.11. Les lignes horizontales que nous pouvons remarquer sont le résultat de l'application du MMC sur chaque ligne séparément. Nous pouvons éventuellement regrouper les informations du voisinage de chaque pixel dans la recherche d'un indice du plus bas coût afin d'obtenir un meilleur résultat.



avec vérification sur les petites images, seuil : 0, taille : 3

après MMC, taux d'erreur des pixels visibles : 17,89%

Figure 5.11 Comparaisons des résultats sans ou avec MMC sur les images de « *Art* »

Ici les matrices de transition et d'émission sont créées à partir de la carte de disparité de référence et une carte de disparité initiale sans interpolation. Nous pouvons obtenir statistiquement des matrices de transition et d'émission pour certains types d'images. Ensuite nous employons ces matrices pour le MMC.

Les pixels noirs de la carte de disparité de gauche dans la Figure 5.11 sont le résultat de la vérification de consistance qui invalide des valeurs de disparités en y mettant 0. Les lignes horizontales de la carte de disparité de droite sont dues à la méthode que nous avons employée, qui fait la recherche de la meilleure séquence ligne par ligne, sans prendre en compte des informations voisines en haut et en bas.

La Figure 5.12 montre les résultats de MMC avec une matrice de transition obtenue statistiquement ou créée avec un paramètre en employant l'équation 5.3. La taille de la fenêtre de comparaison est 3.

MMC avec matrice de transition
obtenue statistiquement,
taux d'erreur des pixels visibles : 17,89%

MMC avec matrice de transition
créée avec un paramètre $\sigma = 10^6$,
matrice d'émission apprise,
taux d'erreur des pixels visibles : 19,87%

une passe sur les petites images,
sans remplissage de disparités HD,
MMC avec matrices de transition
et d'émission par paramètre $\sigma = 10^6$,
taux d'erreur des pixels visibles : 36,62%

avec remplissage de disparités HD,
MMC avec matrices de transition
et d'émission par paramètre $\sigma = 10^6$,
taux d'erreur des pixels visibles : 30,20%

Figure 5.12 Comparaison des résultats de MMC avec la matrice de transition obtenue statistiquement ou créée à partir d'un paramètre sur les images de « *Art* »

Pour ces images de « *Art* », nous pouvons constater à pouvoir simplement employer un paramètre pour définir une matrice de transition, sans sacrifier beaucoup la qualité obtenue. Le principe d'employer un paramètre est de supposer qu'une disparité a plus de tendances de conserver sa valeur dans son voisin à droite. Nous pouvons aussi établir d'autres règles. Une

approche simple que nous avons montrée peut bien fonctionner dans la plupart de régions si nous ignorons les détails. Cela a aussi indiqué que nous pouvons créer une matrice de transition pour chacun de certains types d'images, et l'employer dans certaines applications désignées.

Si nous employons le MMC avec la matrice de transition en paramètres, le résultat obtenu peut perdre certaines informations. Dans ce cas, nous pouvons employer les disparités initialement valides pour remplacer une partie des résultats afin de retrouver des détails, surtout aux bordures d'objet. Dans ce dernier cas, il faut employer certaines méthodes pour identifier les pixels utiles. Nous avons remarqué que la fiabilité des disparités initiales n'est pas très élevée même après la vérification de consistance. Compte tenu de coût élevé de MMC, dans certaines occasions, il va mieux d'identifier des zones difficiles à traiter, de trouver des repères fiables et d'employer l'interpolation pour créer une carte de disparité.

Nous pouvons aussi employer des informations du voisinage de chaque pixel afin d'augmenter la pertinence de chaque disparité. Cela devient le problème de programmation dynamique. Cette solution ressemblera beaucoup à la méthode semi-globale qui combine les coûts du voisinage dans la recherche de l'indice du plus bas coût.

Ce modèle peut être appliqué sur d'autres images avec l'environnement similaire et la carte de disparité similaire. Malheureusement à la limite de notre ensemble d'images et de cartes de disparité, nous n'avons pas pu produire un exemple satisfaisant sans avoir recours à la carte de disparité de référence correspondante.

5.5.3 Plus d'une exécution de MMC

Parce que le MMC traite des pixels d'une ligne pixel par pixel horizontalement, le résultat à droite de la Figure 5.11 montre souvent de bandes horizontales parce que chaque ligne est indépendante dans cette méthode. Nous pouvons passer un autre MMC verticalement ou même encore un MMC horizontal ou HMM vertical des directions inverses pour améliorer la qualité. Nous pouvons traiter en plusieurs étapes comme le montre la Figure 5.13 où les lignes noires solides font partie d'un modèle horizontal de gauche à droite, les lignes bleues pointillées sont venues d'un modèle vertical de haut en bas, les lignes rouges solides font partie d'un modèle horizontal de droite à gauche, tandis que les lignes vertes pointillées font partie d'un modèle horizontal de bas en haut.

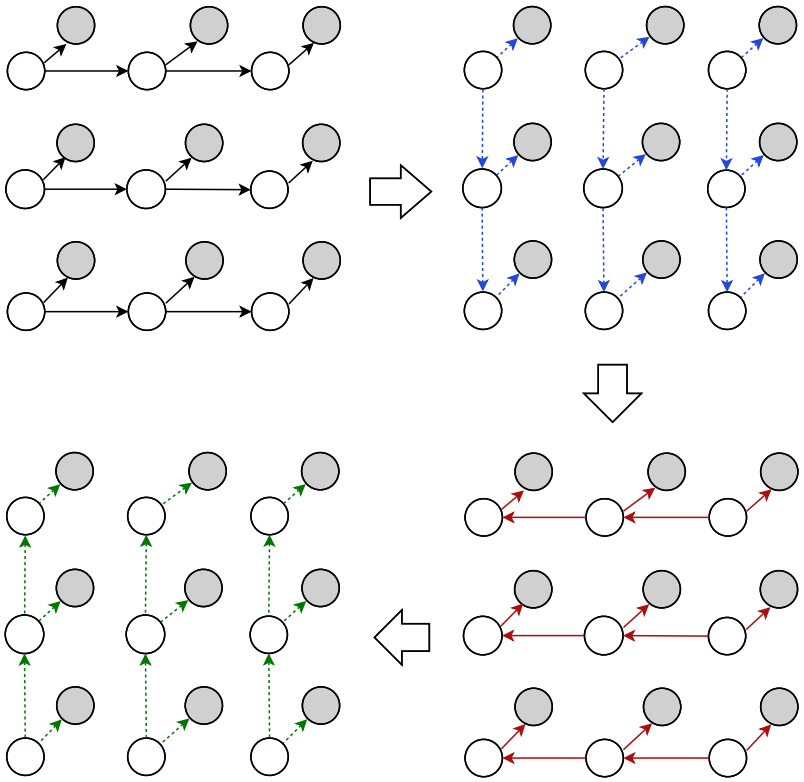


Figure 5.13 Généralisation d'un modèle de Markov caché

Nous pouvons également traiter en deux étapes comme le modèle dans la Figure 5.14 où les les lignes noires solides font partie d'un modèle horizontal tandis que lignes bleues pointillées sont venues d'un modèle vertical.

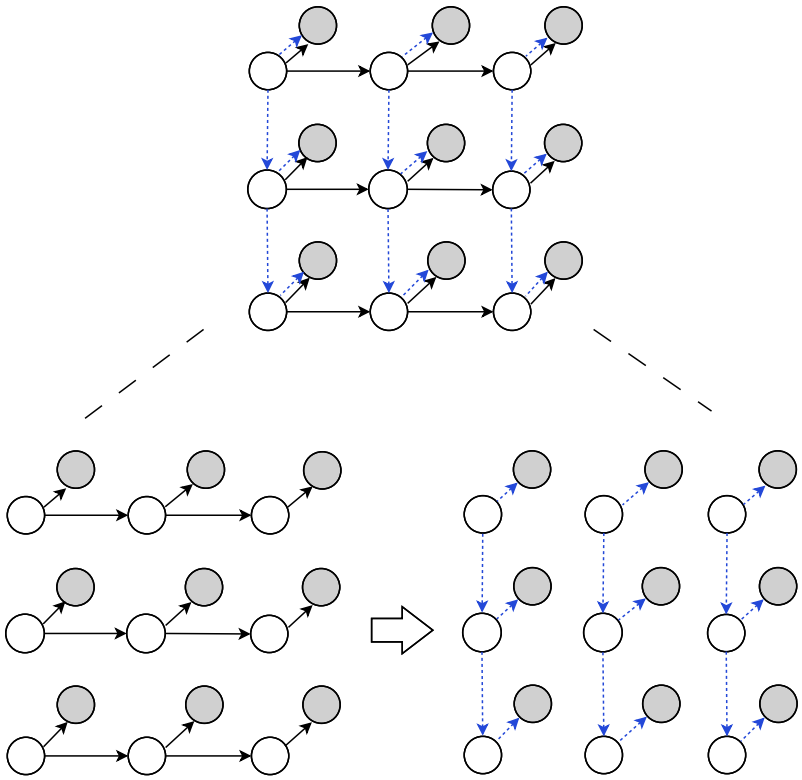


Figure 5.14 Une généralisation d'un modèle de Markov caché en deux étapes

La Figure 5.15 montre les résultats de deux ou trois passes de MMC avec une matrice de transition obtenue statistiquement. La taille de la fenêtre de comparaison est 3.

Pour étudier l'impact de deux ou plusieurs passes de MMC avec une matrice de transition obtenue statistiquement, nous avons créé un Tableau 5.5. Durant ces tests, nous avons employé la Méthode 2, avec la disparité dans la gamme [0, 79] et une taille de fenêtre 3. Les images employées sont les petites et grandes images de la taille 463×370 et 1390×1110 de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005).

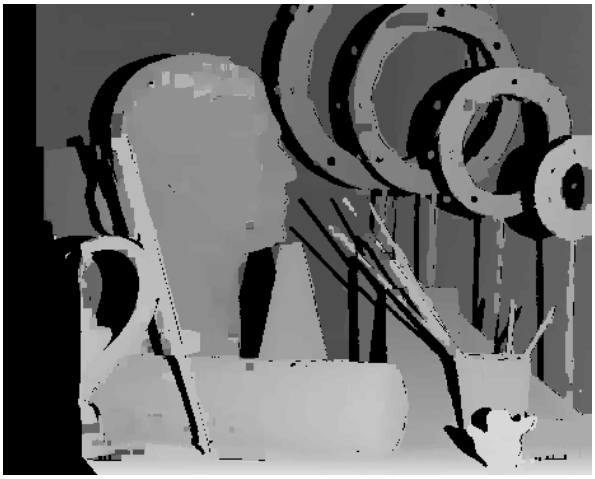
Tableau 5.5 Comparaisons des performances avec Viterbi en employant les images de « *Art* »

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur
Seuil de double vérification : 0		
Avec les petites images de 463 × 370 seulement		
Avec la vérification de consistance sur les petites images + interpolation	39,73%	49,11%
Après une passe horizontale de MMC	17,89%	24,01%
Une passe horizontale et une autre verticale de MMC	10,68%	15,93%
Une passe horizontale, une passe verticale et une dernière passe horizontale de MMC de droite à gauche	10,38%	15,66%
Une passe horizontale, une passe verticale et une passe horizontale de droite à gauche et une dernière passe de bas en haut de MMC	10,32%	15,64%
Avec les petites et grandes images de 463 × 370 et de 1390 × 1110 + remplissage		
Avec la vérification de consistance sur les petites et grandes images + remplissage HD + interpolation	34,14%	44,43%
Après une passe horizontale de MMC	16,63%	23,86%
Une passe horizontale et une autre verticale de MMC	10,45%	16,55%
Une passe horizontale, une passe verticale et une dernière passe horizontale de MMC de droite à gauche	10,35%	16,40%
Une passe horizontale, une passe verticale et une passe horizontale de droite à gauche et une dernière passe de MMC de bas en haut	10,22%	16,32%

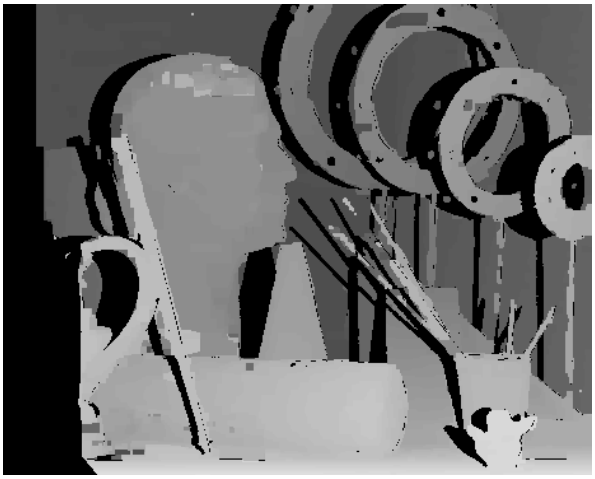
La Figure 5.15 montre les cartes de disparité obtenues avec la vérification de consistance sur les petites et grandes images de 463 × 370 et de 1390 × 1110, le remplissage et plusieurs passes de MMC.

avec vérification sur les petites et grandes images + interpolation, taille : 3

après une passe horizontale de MMC



MMC horizontal + MMC vertical (deux passes de MMC)



MMC horizontal + MMC vertical + MMC MMC horizontal droite-gauche + MMC vertical bas-haut (quatre passes)

Figure 5.15 Comparaison des résultats de remplissage, d'une, de deux et de quatre passes de MMC avec matrices de transition et d'émission obtenues statistiquement sur les images de « *Art* »

Nous pouvons remarquer une constante amélioration de qualité. Pourtant, après deux passes, l'amélioration sera limitée compte tenu la nature de l'algorithme Viterbi qui suive le chemin maximal.

5.5.4 Performance

Pour réduire le temps d'exécution avec le MMC, nous pouvons réduire la taille des deux matrices de transition et d'émission, nous pouvons ignorer les disparités déjà observées, les

utiliser comme vraies disparités.

Nous pouvons réduire le nombre de symboles d'observation M. Après une vérification partielle, nous avons observé que si la taille de l'ensemble de symboles d'observation M est réduite, on peut réduire le temps d'exécution, mais de façon limitée. Il est clair que si la taille de l'ensemble d'états N est réduite, le temps d'exécution peut lui aussi être réduit. Le nombre d'états N est le facteur principal de la réduction du temps de calcul.

Nous pouvons écrire un programme avec un N de valeur réduite. Nous pouvons aussi réduire le nombre de fils d'exécution dans un bloc selon la particularité d'une application, par exemple, selon la taille de la carte de disparité recherchée. Nous pouvons espérer que le temps en GPGPU soit réduit à une valeur raisonnable pour les cartes de certaines tailles.

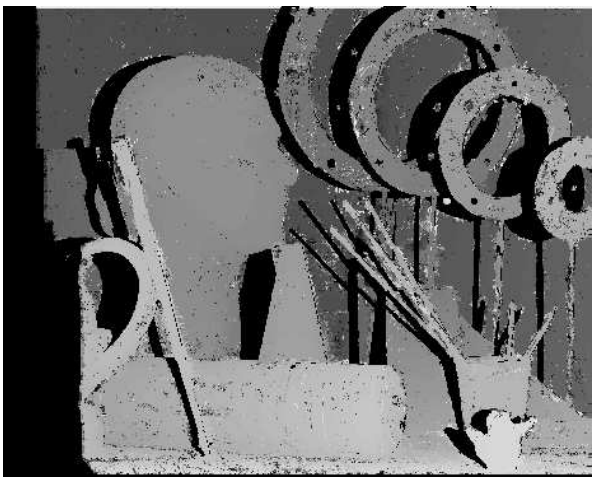
5.6 Combinaisons de Census et de MMC

Nous croyons pouvoir supposer que l'observation initiale influence de beaucoup le résultat d'un MMC. Pour confirmer cette hypothèse, nous avons employé la combinaison de Census et de MMC, avec la disparité dans la gamme [0, 79] et une taille de fenêtre 15. Les images employées sont les petites et grandes images de la taille 463×370 et 1390×1110 de « *Art* » de Middlebury présentées par Blasiak *et al.* (voir Blasiak *et al.*, 2005).

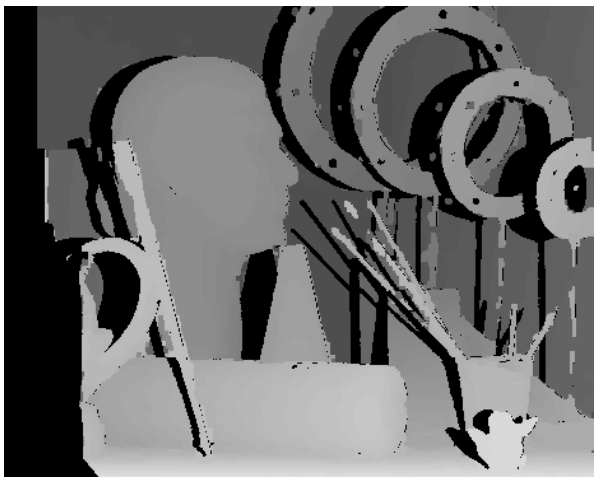
Tableau 5.6 Comparaisons des taux d'erreur en employant la combinaison de Census et de MMC sur les images de « *Art* »

Méthode	Taux d'erreur des pixels visibles	Taux d'erreur
Seuil de double vérification : 0		
Avec les petites images de 463 × 370 seulement		
Une seule passe avec les petites images + interpolation	22,42%	40,62%
Après une passe horizontale de MMC	12,11%	24,20%
Une passe horizontale et une autre verticale de MMC	8,97%	17,31%
Une passe horizontale, une passe verticale et une dernière passe horizontale de MMC de droite à gauche	8,84%	16,94%
Une passe horizontale, une passe verticale et une passe horizontale de droite à gauche et une dernière passe de bas en haut de MMC	8,83%	16,89%
Avec les petites et grandes images de 463 × 370 et de 1390 × 1110 + remplissage		
Avec la vérification de consistance sur les petites, + remplissage HD d'une seule passe sur les grandes images + interpolation	15,64%	35,04%
Après une passe horizontale de MMC	7,32%	19,00%
Une passe horizontale et une autre verticale de MMC	5,18%	13,30%
Une passe horizontale, une passe verticale et une dernière passe horizontale de MMC de droite à gauche	5,10%	12,97%
Une passe horizontale, une passe verticale et une passe horizontale de droite à gauche et une dernière passe de bas en haut de MMC	5,08%	12,88%

La Figure 5.16 montre les cartes de disparité obtenues avec la vérification de consistance en employant les petites de 463 × 370 et le remplissage d'une seule passe avec les grandes images de 1390 × 1110 et puis l'interpolation ou quatre passes de MMC.



avec vérification en employant les petites images et une passe avec les grandes images+interpolation

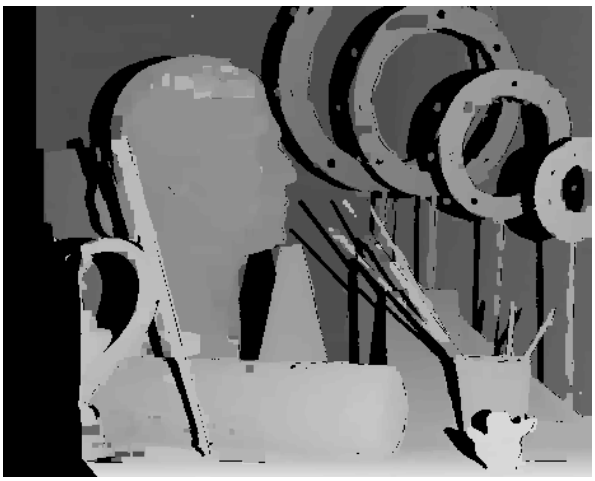


MMC horizontal + MMC vertical + MMC horizontal droite-gauche + MMC vertical bas-haut (quatre passes)

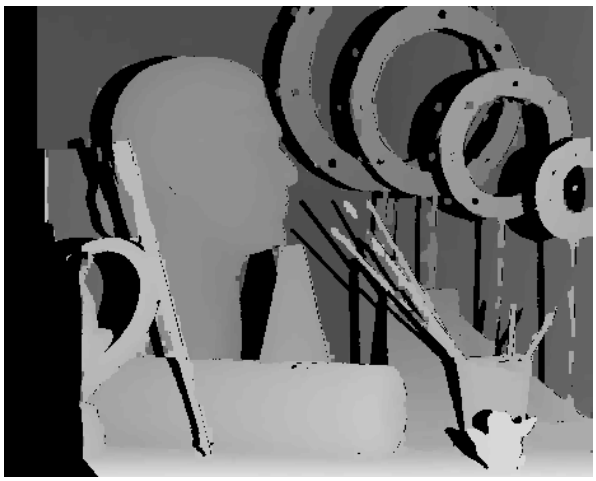
Figure 5.16 Comparaison des résultats de Census puis vérification de consistance sur les petites images de « *Art* », remplissage d'une seule passe sur les grandes images, et de quatre passes de MMC avec matrices de transition et d'émission obtenues statistiquement

Nous pouvons voir que la combinaison de Census, de vérification de consistance, de remplissage et de quatre passes de MMC (MMC horizontal, MMC vertical, MMC horizontal droite-gauche et le MMC vertical bas-haut) a un bon résultat. Pour connaître plus sur d'autres images, nous avons comparé la combinaison de mise en correspondance par bloc (BM), de vérification de consistance, de remplissage et de quatre passes de MMC avec la combinaison de Census, de vérification de consistance sur les petites cartes de disparité en employant les petites images, de remplissage d'une passe en employant les grandes images et de quatre passes de MMC.

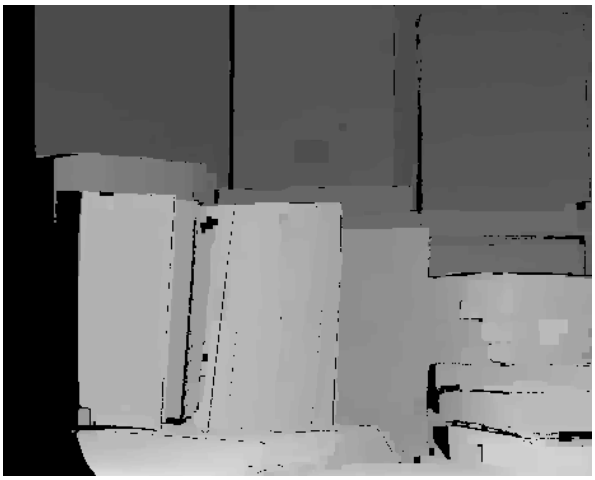
La Figure 5.17, la Figure 5.18 et la Figure 5.19 montrent les résultats de combinaison de MMC et de la mise en correspondance par bloc (BM) ou de Census sur toutes les images de Middlebury présentées par Blasiak *et al.*. (voir Blasiak *et al.*, 2005). La taille de fenêtre de la mise en correspondance par bloc (BM) est 3. La taille de fenêtre de Census pour les petites et grandes images est 15. Nous avons enlevé les pixels invisibles en les ramenant à la valeur 0. Nous n'avons appliqué que la vérification de consistance, le remplissage de disparités des images en plus haute définition et quatre passes de MMC (MMC horizontal de gauche à droite, MMC vertical de haut en bas, MMC horizontal de droite à gauche et MMC vertical de bas en haut).



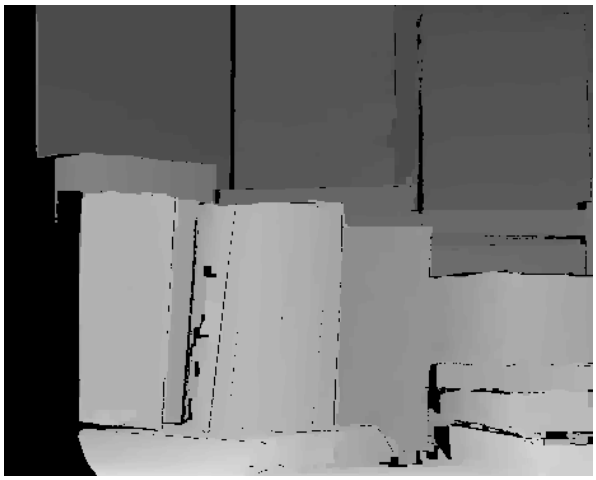
basé sur BM, taux d'erreur : 10,22%



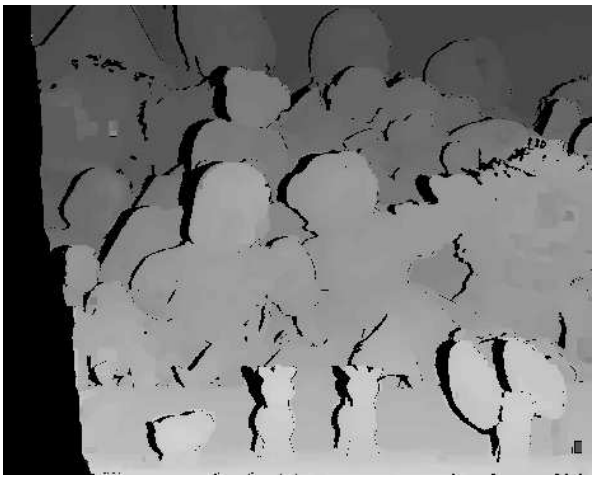
basé sur Census, taux d'erreur : 5,08%



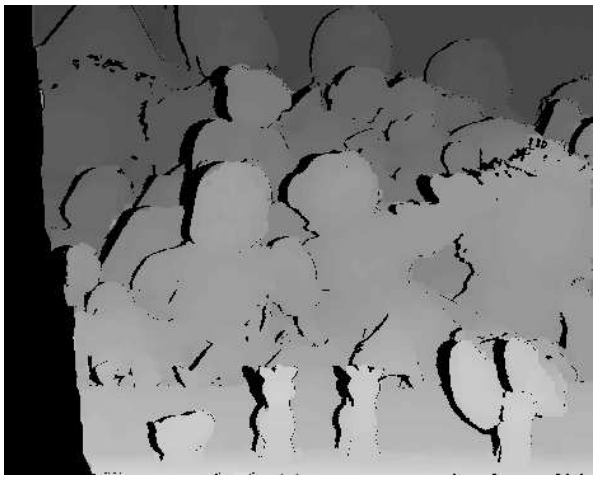
basé sur BM, taux d'erreur : 10,81%



basé sur Census, taux d'erreur : 4,50%



basé sur BM, taux d'erreur : 8,23%

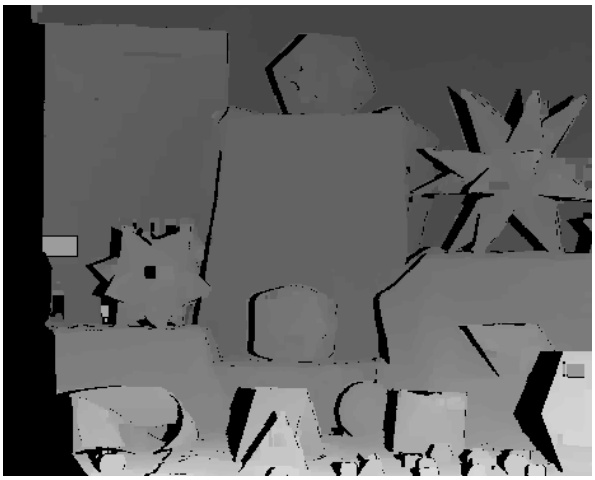


basé sur Census, taux d'erreur : 2,88%

Figure 5.17 Comparaisons des cartes obtenues basées sur la combinaison de MMC et de la mise en correspondance par bloc ou de Census, avec la vérification de consistance, le remplissage de disparités des images en plus haute définition

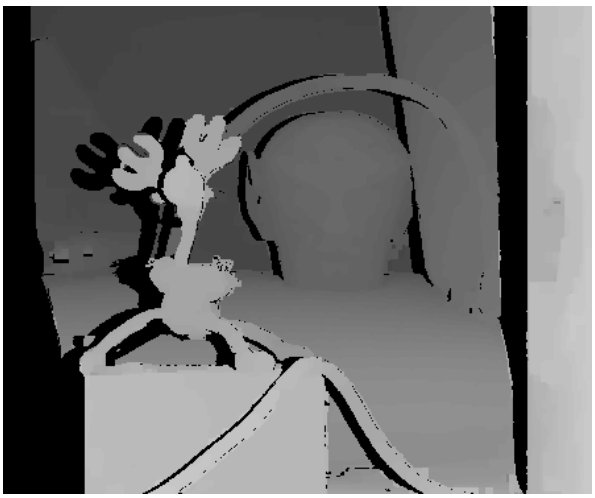
basé sur BM, taux d'erreur : 21,49%

basé sur Census, taux d'erreur : 9,13%

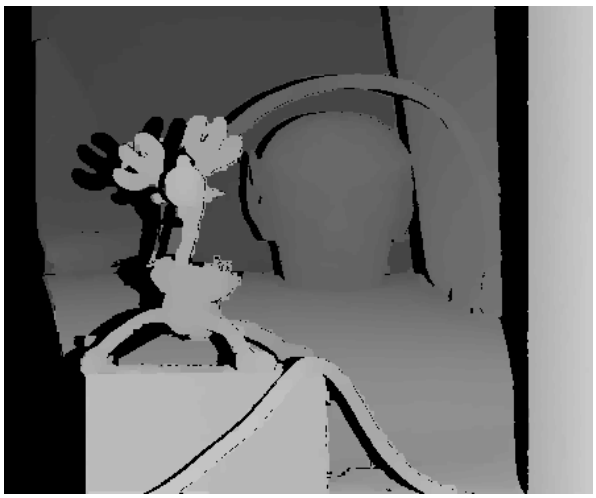


basé sur BM, taux d'erreur : 6,64%

basé sur Census, taux d'erreur : 3,98%

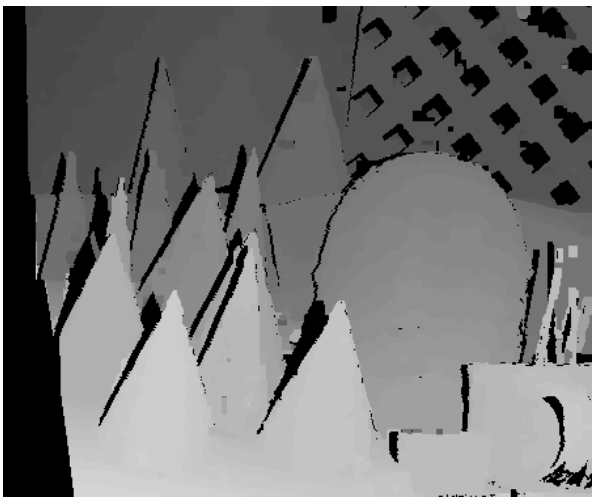


basé sur BM, taux d'erreur : 7,40%

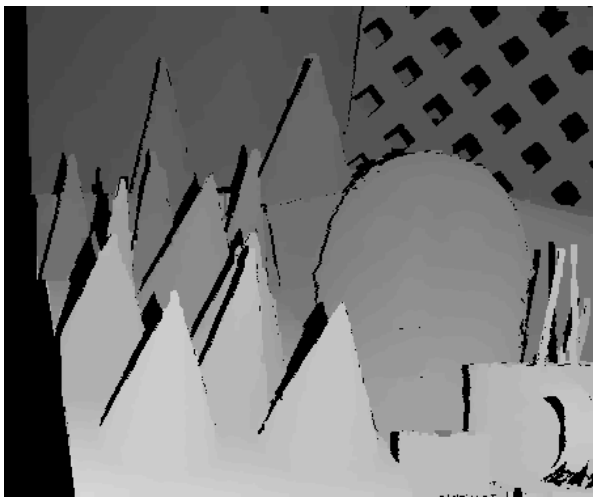


basé sur Census, taux d'erreur : 2,31%

Figure 5.18 Comparaisons des cartes obtenues basées sur la combinaison de MMC et de la mise en correspondance par bloc ou de Census, avec la vérification de consistance, le remplissage de disparités des images en plus haute définition

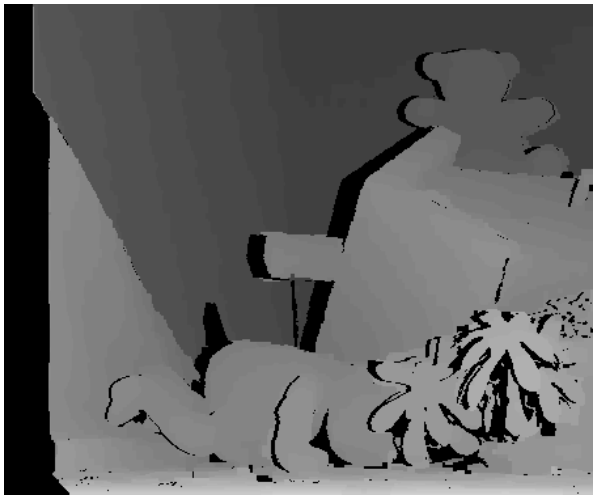


basé sur BM, taux d'erreur : 4,95%



basé sur Census, taux d'erreur : 2,44%

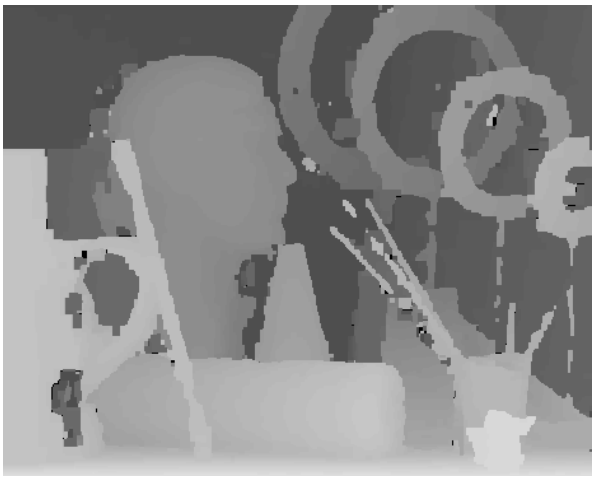
basé sur BM, taux d'erreur : 7,89%



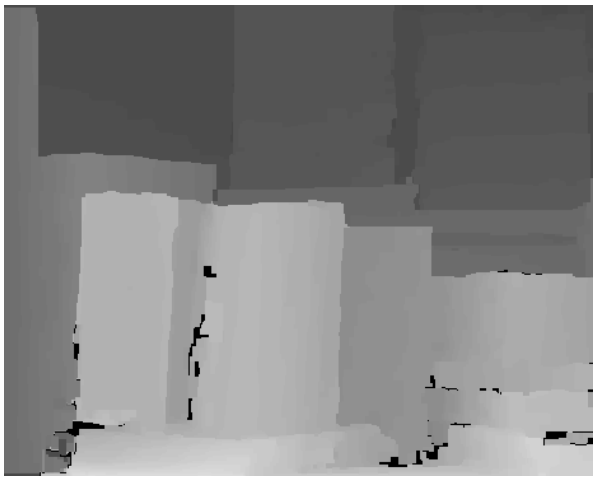
basé sur Census, taux d'erreur : 3,86%

Figure 5.19 Comparaisons des cartes obtenues basées sur la combinaison de MMC et de la mise en correspondance par bloc ou de Census, avec la vérification de consistance, le remplissage de disparités des images en plus haute définition

Pour le cas de la combinaison de Census, de vérification de consistance sur les cartes de disparité en employant les petites images, de remplissage d'une seule passe de grande carte de disparité en employant les grandes images et de quatre passes de MMC, nous voulons aussi connaître la carte complète au lieu des zones visibles. La Figure 5.20 et la Figure 5.21 montrent quelques cartes de disparité avec le taux d'erreur global.

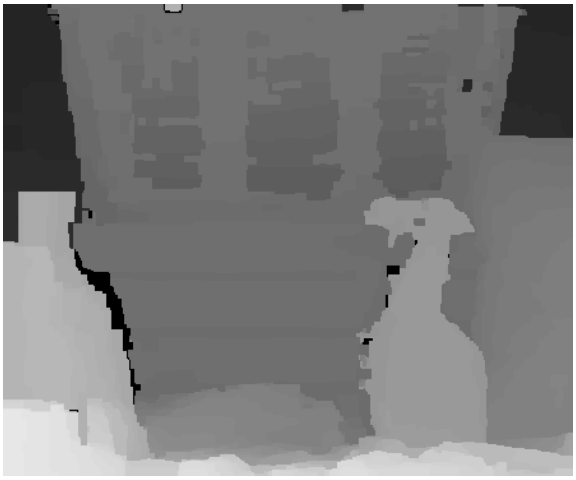


Art, taux d'erreur global : 12,88%



Books taux d'erreur global : 9,55%

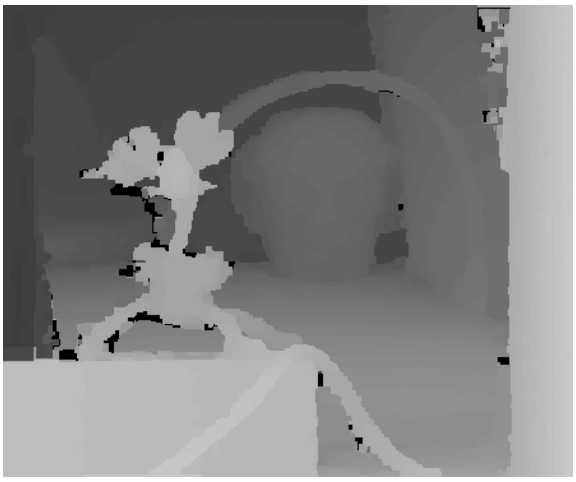
Dolls, taux d'erreur global : 10,34%



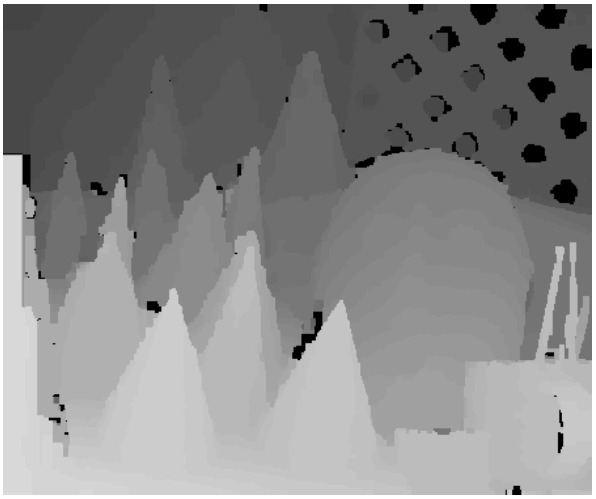
Laundry, taux d'erreur global : 12,80%

Figure 5.20 Comparaisons des cartes obtenues basées sur la combinaison de Census, avec la vérification de consistance, le remplissage de disparités des images en plus haute définition et de MMC

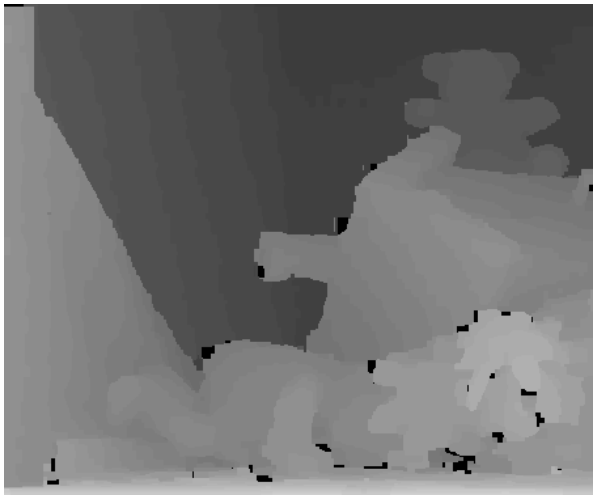
Moebius, taux d'erreur global : 9,88%



Reindeer, taux d'erreur global : 10,42%



Cones, taux d'erreur global : 9,08%



Teddy, taux d'erreur global : 8,73%

Figure 5.21 Comparaisons des cartes obtenues basées sur la combinaison de Census, avec la vérification de consistance, le remplissage de disparités des images en plus haute définition et de MMC

Nous pouvons voir que les cartes obtenues ont une bonne qualité, à l'exception des petits points invalides, ces derniers points peuvent être très bien traités avec la méthode d'enlèvement de taches et l'interpolation MoyenPlusPropagationDuFond afin d'encore améliorer la qualité.

5.7 Comparaisons par chiffre de quelques méthodes

Nous avons dans le Tableau 5.7 des comparaisons de quelques méthodes avec la disparité dans la gamme [0, 79] sur plusieurs petites images de la taille 463×370 et 447×370 de 2005 de Middlebury, la disparité dans la gamme [0, 63] sur les petites images de la taille 450×375 de 2003 de Middlebury, et les grandes images qui sont r ($r \in [3, 4]$) fois plus grandes que les petites images présentées par Scharstein *et al.* (voir Scharstein *et al.*, 2003) et par Blasiak *et al.* (voir Blasiak *et al.*, 2005). μ signifie la moyenne des taux d'erreur des pixels visibles. Les tests sont effectués sur NVIDIA® GeForce® GTX 580 et Intel® Core™ i7-2600K @ 3,40 GHz sous Windows 7 Professional.

Si on n'emploie que les petites images, fait une seule passe de gauche à droite puis l'interpolation,

- La méthode BM utilise la mise en correspondance par bloc (BM) Méthode 2 d'une seule passe de gauche à droite et l'interpolation. La taille de fenêtre est 5.
- La méthode Census emploie Census d'une seule passe de gauche à droite et l'interpolation. La taille de fenêtre est 15.
- La méthode « βBM+Census^2 » emploie la combinaison de Census et de mise en correspondance par bloc (BM) d'une seule passe de gauche à droite et l'interpolation. La taille de fenêtre est 15 pour la partie Census et 3 pour la partie BM. $\beta = 0,6$.

Si on n'emploie que les petites images, fait la vérification de consistance et l'interpolation,

- La méthode BM utilise la mise en correspondance par bloc (BM) Méthode 2, la vérification de consistance et l'interpolation. La taille de fenêtre est 5.
- La méthode Census emploie Census, la vérification de consistance et l'interpolation. La taille de fenêtre est 15.

Si on n'emploie que les petites images, fait la vérification de consistance, l'enlèvement de tache et l'interpolation, la taille de fenêtre pour la méthode BM est 5.

Si on n'emploie que les grandes images avec le ratio r ($r \in [3, 4]$) entre la taille des grandes images et celle des petites images,

- La méthode BM utilise la mise en correspondance par bloc (BM) Méthode 2 d'une seule passe de gauche à droite, ensuite réduit la carte de disparité obtenue à l'échelle de $1/r$ et fait l'interpolation. La taille de fenêtre est $5r$ ($r \in [3, 4]$).

- La méthode Census(19) emploie Census d'une seule passe de gauche à droite, réduit à l'échelle de $1/r$ ($r \in [3,4]$) et l'interpolation. La taille de fenêtre est 19.
- La méthode « βBM+Census^2 » utilise une combinaison de Census et de mise en correspondance par bloc d'une seule passe et l'interpolation. La taille de fenêtre de comparaison de la mise en correspondance par bloc pour les grandes images est $3r$ ($r \in [3,4]$). La taille de fenêtre de comparaison de Census est 15. $\beta = 0,2$.

Si on emploie les petites et grandes images avec le ratio r ($r \in [3,4]$) entre la taille des grandes images et celle des petites images,

- La méthode BM utilise la mise en correspondance par bloc (BM) Méthode 2 avec la vérification de consistance sur les petites images, le remplissage de disparités d'une passe des grandes images et l'interpolation. La taille de fenêtre est 5 pour les petites images, celle pour les grandes images est en proportion avec la taille d'images, soit $5r$ ($r \in [3,4]$).
- La méthode Census utilise Census avec la vérification de consistance sur les petites images, le remplissage de disparités d'une passe des grandes images et l'interpolation. La taille de fenêtre est 15 tant pour les petites images que pour les grandes images.
- La méthode « Census $\triangleright$BM » utilise une séquence de Census et de la mise en correspondance par bloc. Nous traitons les petites images en employant Census avec la vérification de consistance. Puis nous faisons le remplissage de disparités d'une passe des grandes images en employant Census, puis nous faisons encore une fois le remplissage en employant les grandes images avec la mise en correspondance par bloc d'une seule passe de gauche à droite avec la taille de fenêtre de comparaison de $5 \times r$. Comme la dernière étape, nous faisons l'interpolation. La taille de fenêtre de Census tant pour les petites images que pour les grandes images est 15.
- La méthode « βBM+Census^2 » utilise une combinaison de Census et de mise en correspondance par bloc avec la vérification de consistance sur les petites images, le remplissage de disparités d'une passe des grandes images et l'interpolation. La taille de fenêtre de la partie de la mise en correspondance par bloc pour les petites images est 3, celle pour les grandes images est proportionnelle avec la taille d'images, soit $3r$ ($r \in [3,4]$). La taille de fenêtre de la partie Census est 15. Pour les petites images : $\beta = 0,6$. Pour les grandes images : $\beta = 0,2$.
- La méthode « Census $\triangleright\beta$BM+Census^2 » utilise Census avec la vérification de consistance sur les petites images, le remplissage de disparités d'une passe d'une combinaison de Census et de mise en correspondance par bloc sur les grandes images et l'interpolation. La taille de fenêtre pour les grandes images est $3r$ ($r \in [3,4]$), proportionnelle avec la taille d'images. La taille de fenêtre tant pour Census et pour la partie Census

de la combinaison de Census et de mise en correspondance par bloc est 15. Pour la combinaison de Census et de mise en correspondance par bloc : $\beta = 0,2$.

Si on emploie les petites images et passe au MMC pour une interférence semi-globale avec les matrices de transition et d'émission définies par un groupe de paramètres identiques $\sigma = 10^6$ en employant les équations 5.3 et 5.4,

- La méthode CensusMMC2 emploie une combinaison de deux passes de MMC et d'une seule passe de Census sur les petites images. La taille de fenêtre de comparaison est 15.
- La méthode CensusMMC4 emploie une combinaison de quatre passes de MMC et d'une seule passe de Census sur les petites images. La taille de fenêtre de comparaison est 15.
- La méthode βBM+Census^2+MMC2 emploie une combinaison de Census et de mise en correspondance par bloc d'une seule passe sur les petites images, $\beta = 0,6$, et ensuite fait deux passes de MMC. La taille de fenêtre de comparaison de Census est 15, celle de la mise en correspondance par bloc est 3.
- La méthode βBM+Census^2+MMC4 emploie une combinaison de Census et de mise en correspondance par bloc d'une seule passe sur les petites images, $\beta = 0,6$, et ensuite fait quatre passes de MMC. La taille de fenêtre de comparaison de Census est 15, celle de la mise en correspondance par bloc est 3.

Si on emploie les petites et grandes images et passe au MMC pour une interférence semi-globale avec les matrices de transition et d'émission définies par un groupe de paramètres identiques $\sigma = 10^6$ en employant les équations 5.3 et 5.4.

- La méthode BM-MMC2 emploie une combinaison de deux passes de MMC et de la mise en correspondance par bloc avec la vérification de consistance sur les petites images, une seule passe sur les grandes images et le remplissage. La taille de fenêtre de comparaison est 5 pour les petites images. Celle pour les grandes images est $5r$ ($r \in [3,4]$).
- La méthode BM-MMC4 emploie une combinaison de quatre passes de MMC et de la mise en correspondance par bloc avec la vérification de consistance sur les petites images, une seule passe sur les grandes images et le remplissage. La taille de fenêtre de comparaison est 5 pour les petites images. Celle pour les grandes images est $5r$ ($r \in [3,4]$).
- La méthode CensusMMC2 emploie une combinaison de deux passes de MMC et de la Census avec la vérification de consistance sur les petites images et le remplissage de disparités d'une seule passe des grandes images. La taille de fenêtre de comparaison est 15.
- La méthode CensusMMC4 emploie une combinaison de quatre passes de MMC et de la Census avec la vérification de consistance sur les petites images et le remplissage de disparités d'une seule passe des grandes images. La taille de fenêtre de comparaison est

15.

Nous avons observé que pour la plupart de cas, chaque pixel de disparité a tendance à conserver son état actuel. $P(d_i = j|d_i = j), 1 \leq i, j \leq N$ sera plus élevée que les autres probabilités en général. Supposons que les disparités changent graduellement, chaque voisin direct d'un pixel de disparité a aussi une probabilité élevée, soit $P(d_{i+1} = j - 1|d_i = j), 1 \leq i \leq N - 1, 2 \leq j \leq N$ et $P(d_{i+1} = j + 1|d_i = j), 1 \leq i, j \leq N - 1$ sont aussi élevées. Si nous supposons encore que chaque autre état a la possibilité égale d'arriver à n'importe quel état, nous avons une matrice de transition estimée avec de voisins directs en employant d'autres cartes de disparité :

$$\begin{cases} P(d_i = j|d_i = j) = \alpha & 1 \leq i, j \leq N, \alpha > 0 \\ P(d_{i+1} = j - 1|d_i = j) = \beta & 1 \leq i \leq N - 1, 2 \leq j \leq N, \beta > 0 \\ P(d_{i+1} = j + 1|d_i = j) = \gamma & 1 \leq i \leq N - 1, 1 \leq j \leq N - 1, \gamma > 0 \\ P(d_{i+1} < j - 1 \,||\, d_{i+1} > j + 1|d_i = j) = \dfrac{1 - \alpha - \beta - \gamma}{N - k} & |j - i| > 1 \,||\, \alpha = 0 \,||\, \beta = 0 \,||\, \gamma = 0, \\ & 1 \leq i, j \leq N, \\ & k \text{ est le nombre de } \alpha > 0, \beta > 0, \gamma > 0 \end{cases} \tag{5.5}$$

D'une manière similaire, nous pouvons estimer la matrice d'émission.

Dans ce mémoire, quand on parle d'autres cartes de disparité, pour les images de Middlebury de 2005, nous signifions d'autres images à l'exception de celles employées actuellement. Par exemple, pour les images de « *Art* », les autres images incluent les images de « *Books* », de « *Dolls* », de « *Laundry* », de « *Moebius* » et de « *Reindeer* ». D'une manière similaire, pour les images de Middlebury de 2003, les autres images pour « *Cones* » sont les images de « *Teddy* », et vice versa. La raison est due à la différence des disparités maximales employées qui sont 79 et 63 pour les images de 2005 et celles de 2003 respectivement.

Si on emploie les petites et grandes images et fait une interférence semi-globale avec les matrices de transition et d'émission ayant 3 éléments centraux estimées par d'autres cartes de disparité.

- La méthode BM-MMC2 emploie une combinaison de deux passes de MMC et de la mise en correspondance par bloc avec la vérification de consistance sur les petites images, une seule passe sur les grandes images et le remplissage. La taille de fenêtre de comparaison est 5 pour les petites images. Celle pour les grandes images est $5r$ ($r \in [3, 4]$).
- La méthode CensusMMC2 emploie une combinaison de deux passes de MMC et de la Census avec la vérification de consistance sur les petites images et le remplissage de disparités d'une seule passe des grandes images. La taille de fenêtre de comparaison est

15.

Si on emploie les petites et grandes images et fait une interférence semi-globale avec matrices apprises selon les cartes de disparité de référence.

- La méthode BM(3)-MMC2 emploie une combinaison de deux passes de MMC et de la mise en correspondance par bloc avec la vérification de consistance sur les petites et grandes images et le remplissage. La taille de fenêtre de comparaison est 3 pour les petites images. Celle pour les grandes images est $3r$ ($r \in [3, 4]$).
- La méthode BM(3)-MMC4 emploie une combinaison de quatre passes de MMC et de la mise en correspondance par bloc avec la vérification de consistance sur les petites et grandes images et le remplissage. La taille de fenêtre de comparaison est 3 pour les petites images. Celle pour les grandes images est $3r$ ($r \in [3, 4]$).
- La méthode CensusMMC2 emploie une combinaison de deux passes de MMC et de la Census avec la vérification de consistance sur les petites images et le remplissage de disparités d'une seule passe des grandes images. La taille de fenêtre de comparaison est 15.
- La méthode CensusMMC4 emploie une combinaison de quatre passes de MMC et de la Census avec la vérification de consistance sur les petites images et le remplissage de disparités d'une seule passe sur les grandes images. La taille de fenêtre de comparaison est 15.

Les données d'interférence semi-globale nous donnent une idée de la limite de cette méthode, soit le mieux que nous pouvons faire.

Tableau 5.7 Taux d'erreur des pixels visibles en pourcentage (%) des quelques combinaisons

Méthode	μ	*Art*	*Books*	*Dolls*	*Laundry*	*Moebius*	*Reindeer*	*Cones*	*Teddy*
Avec les petites images de 463×370, 447×370 ou 450×375									
A. Une seule passe de gauche à droite puis l'interpolation									
BM	27,95	33,76	35,73	17,25	45,50	26,53	25,44	17,54	21,81
Census	19,89	22,42	23,79	12,26	34,34	18,75	20,12	11,27	16,16
βBM+Census^2	15,61	17,58	18,55	8,85	28,62	15,33	15,78	7,30	12,90
B. La vérification de consistance puis l'interpolation									
BM	27,16	32,17	34,54	16,28	46,38	25,49	24,20	17,31	20,92
Census	17,95	20,53	20,57	12,24	31,81	17,95	15,87	10,93	12,69
C. La vérification de consistance, l'enlèvement de tache et l'interpolation									
BM	26,20	30,94	33,24	16,45	43,69	24,73	24,50	16,58	19,44
Avec les grandes images de 1342×1110, 1390×1110 ou 1800×1500									
BM	22,48	27,25	25,46	15,11	39,17	21,63	19,39	13,91	17,94
Census(19)	17,15	16,17	25,49	9,63	31,79	15,57	12,27	10,01	16,26
βBM+Census^2	13,93	12,27	19,94	6,80	28,48	14,16	8,94	7,80	13,03
Avec les petites et grandes images dont la taille est r ($r \in [3, 4]$) fois que celle des petites									
BM	23,87	29,03	28,39	15,55	40,69	22,99	20,63	14,63	19,01
Census	15,34	15,64	22,24	8,76	28,85	14,69	11,38	7,29	13,83
Census $\triangleright$BM	16,06	16,50	23,90	9,52	30,41	15,25	12,09	7,03	13,76
βBM+Census^2	20,69	20,93	29,23	12,42	35,45	19,71	16,36	13,72	17,69
Census $\triangleright$ βBM+Census^2	12,23	12,33	16,32	6,21	25,72	12,90	8,26	5,52	10,59
Interférence semi-globale par MMC avec images $\sim 463 \times 370$ et matrices définies par σ									
CensusMMC2	13,12	16,92	14,41	8,60	26,75	12,76	9,37	7,21	8,97
CensusMMC4	13,10	17,05	14,27	8,68	26,81	12,64	9,36	7,17	8,80
βBM+Census^2+ MMC2	11,60	14,58	12,16	7,33	24,98	11,77	8,37	5,55	8,05
βBM+Census^2+ MMC4	11,58	14,70	12,10	7,37	25,06	11,71	8,26	5,50	7,91
Interférence semi-globale rapprochée par MMC avec les petites et grandes images									
A. Matrices de transition et d'émission définies par paramètre $\sigma = 10^6$									
BM-MMC2	21,77	27,00	24,36	14,55	38,90	19,82	18,29	13,09	18,19
BM-MMC4	21,63	26,95	24,11	14,50	38,90	19,57	18,07	12,91	18,06
CensusMMC2	9,00	10,28	11,57	5,22	19,56	9,23	4,82	4,01	7,28
CensusMMC4	8,95	10,41	11,45	5,27	19,54	9,08	4,81	3,96	7,09
B. Matrices estimées par d'autres cartes de disparité avec 3 éléments centraux									
BM-MMC2	15,20	18,16	14,35	11,90	20,15	14,56	19,95	9,44	13,12
CensusMMC2	7,63	8,52	7,91	4,26	12,64	8,09	4,22	8,04	7,33
C. Matrices apprises avec les cartes de disparité de référence									
BM(3)-MMC2	10,12	10,45	11,09	8,84	22,38	7,06	7,60	5,07	8,46
BM(3)-MMC4	9,70	10,22	10,81	8,23	21,49	6,64	7,40	4,95	7,89
CensusMMC2	4,44	5,18	4,72	2,96	9,15	4,20	2,35	2,51	4,48
CensusMMC4	4,27	5,08	4,50	2,88	9,13	3,98	2,31	2,44	3,86

Pour les cas avec les petites images, nous pouvons remarquer que le cas de la vérification de consistance puis l'interpolation fonctionne mieux que le cas d'une seule passe puis l'interpolation. Cela montre que la vérification de consistance peut éliminer de pixels invalides, l'interpolation MoyenPlusPropagationDuFond peut bien remplir les pixels invalides en employant des informations de voisinage. Si nous faisons de plus l'enlèvement de tache après la vérification de consistance, même si nous avons déjà montré les taux élevés de pixels invalides après la vérification de consistance pour la taille de fenêtre 5 et 3 dans la sous-section 4.8.1 et la sous-section 5.3.3, nous pouvons observer une amélioration de qualité pour le cas de la mise en correspondance par bloc (BM). Nous pouvons possiblement appliquer l'enlèvement de tache avant de faire la vérification de consistance. Compte tenu la nature de l'enlèvement de tache, nous avons raison d'espérer une amélioration de qualité. Nous pouvons aussi passer un filtre BilSub dont l'effet positif a été montré dans la sous-section 4.9.2. Les chiffres ici sont à titre indicatif. Beaucoup de techniques ont été développées pour améliorer la qualité de la mise en correspondance par bloc, nous ne détaillons pas tout dans ce mémoire.

Nous avons employé la taille de fenêtre 19 pour Census dans le cas n'ayant employé que les grandes images parce que si nous employons la même taille de fenêtre de comparaison de 15 comme presque partout ailleurs dans ce mémoire, la méthode Census sur les grandes images fonctionne moins bien que sur les petites images. Une plus grande fenêtre peut normalement améliorer la qualité avec de limites, mais le temps requis sera aussi élevé. Il ne faut pas trop augmenter la taille de fenêtre. Une trop grande de fenêtre de comparaison peut détériorer la qualité.

La mauvaise performance de la méthode « βBM+Census^2 » pour les petites et grandes images avec le remplissage est due à la mauvaise consistance des disparités que nous avons introduite dans la section 5.4, et aussi au fait que nous avons employé une même fonction de combinaison tant pour les petites images que pour les grandes images avec seulement un paramètre pour ajuster, mais cette fonction ne s'adapte pas bien à différentes situations.

Nous pouvons remarquer que la méthode « Census $\triangleright$BM » a un moins bon résultat que la méthode Census, cela peut signifier que notre interpolation fonctionne mieux que les valeurs prises des disparités des grandes images en employant la mise en correspondance par bloc. Cela montre aussi que l'interpolation fonctionne mieux que les valeurs prises des disparités des petites images en employant la mise en correspondance par bloc, parce que nous avons montré que la précision calculée en employant les grandes images est meilleure que celle en employant les petites images, voir la sous-section 4.8.1.

La mauvaise performance de la méthode « Census $\triangleright$BM » contre la méthode « βBM+Census^2 » sur les petites et grandes images montre que la fonction de combinaison rend la carte de disparité de gauche à droite peu cohérente avec la carte de disparité de droite

à gauche parce que l'influence de deux types de coûts provenant de Census et de la mise en correspondance par bloc rend le résultat dépendant de la fonction de combinaison. Malheureusement compte tenu la grande variation de valeur de deux composants de la méthode « Census ▷BM », il est difficile d'obtenir une meilleure fonction de combinaison. Souvent, la fonction de combinaison dépend d'application.

Nous pouvons voir que la méthode CensusMMC4 produit le meilleur résultat. Cette méthode est une combinaison de MMC et de la Census avec la vérification de consistance sur les petites et grandes images et le remplissage de disparités des images en plus haute définition. Nous pouvons aussi remarquer que la qualité après le MMC dépend beaucoup de celle de la carte initiale. Il faut s'assurer la fiabilité de la carte de disparité initiale.

Nous pouvons considérer le taux d'erreur des pixels visibles de CensusMMC4, ou de BM(3)-MMC4 comme la limite supérieure du MMC basé sur Census ou sur BM avec les matrices apprises dans le problème de carte de disparité, tandis que celui avec les matrices définies par un paramètre comme la limite inférieure approximative du MMC. Les cas B montre la possibilité d'amélioration en employant de données réelles avec la stratégie d'apprentissage de tout sauf un à tester.

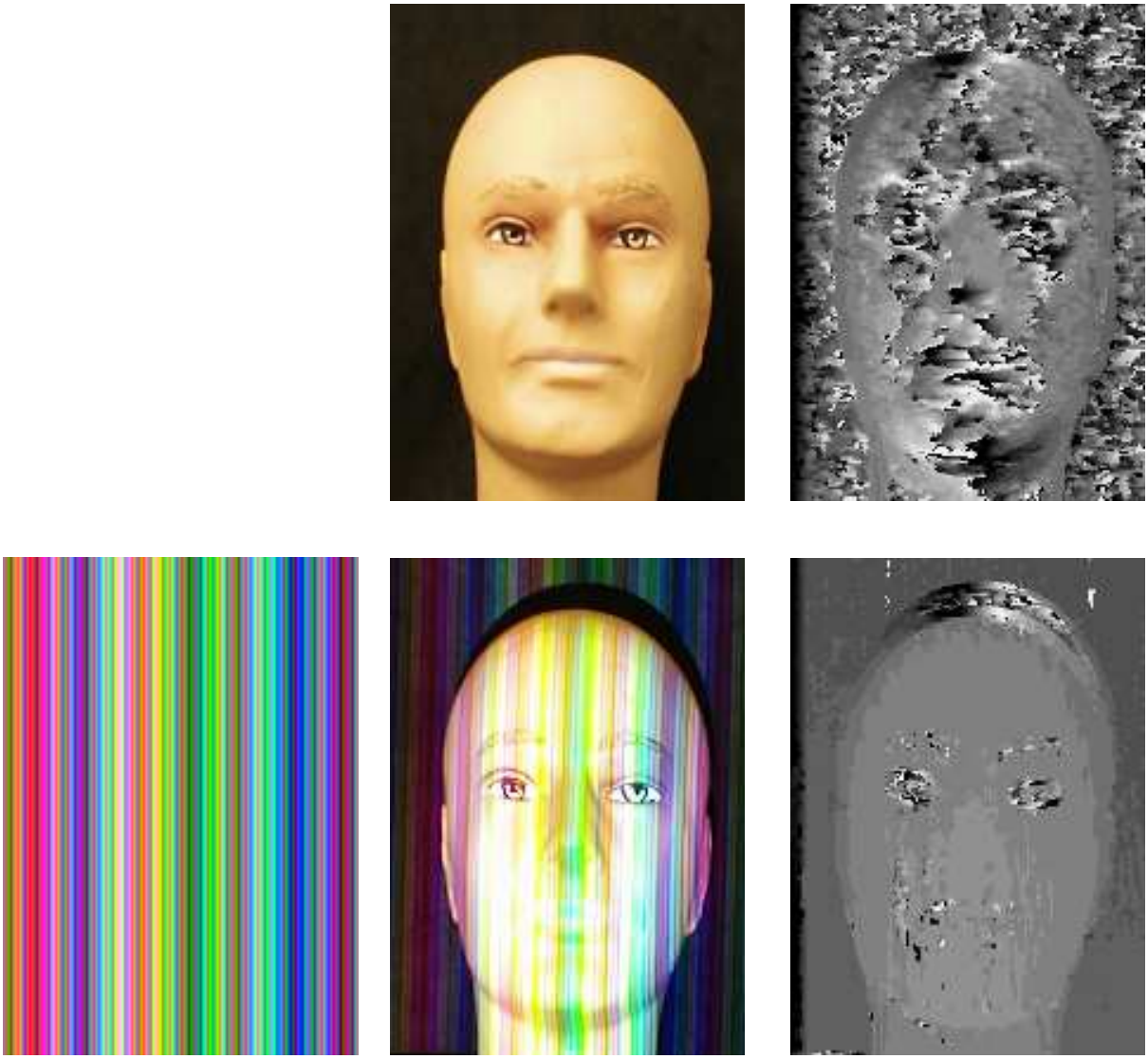
Nous avons listé le temps principal d'exécution en secondes des quelques combinaisons de méthodes dans le Tableau 5.8. Nous n'avons pas d'implémentation de Census et de « βBM+Census^2 » en GPU, en conséquence, nous n'avons listé que le temps de la partie CPU qui nous permet d'avoir une idée d'accélération possible.

Tableau 5.8 Comparaisons de temps principal (en s) des quelques combinaisons de méthodes

Méthode	taille	partie GPU	note GPU	partie CPU	note CPU
BM	463×370	0,003	BM une passe	–	–
Census	463×370	–	–	3,678	Census
βBM+Census^2	463×370	–	–	4,038	βBM+Census^2
Census(19)	1390×1110	–	–	156,769	Census(19)
βBM+Census^2	1390×1110	–	–	184,012	βBM+Census^2
BM	463×370 et 1390×1110	0,076	BM	–	–
Census $\triangleright$BM	463×370 et 1390×1110	0,071	BM pour grandes images	105,416	Census
Census	463×370 et 1390×1110	–	–	103,087	Census
βBM+Census^2	463×370 et 1390×1110	–	–	208,721	βBM+Census^2
Census $\triangleright$ βBM+Census^2	463×370 et 1390×1110	–	–	186,739	Census $\triangleright$ βBM+Census^2
CensusMMC2	463×370	4,673	MMC (2 passes)	3,537	Census
CensusMMC4	463×370	9,467	MMC (4 passes)	3,516	Census
βBM+Census^2+ MMC2	463×370	4,737	MMC (2 passes)	4,012	Census
βBM+Census^2+ MMC4	463×370	9,282	MMC (4 passes)	4,012	Census
BM-MMC2	463×370 et 1390×1110	0,082	BM, HD une passe	–	–
		5,323	MMC (2 passes), matrice de transition sans zéro		
BM-MMC4	463×370 et 1390×1110	0,082	BM, HD une passe	–	–
		10,460	MMC (4 passes), matrice de transition sans zéro		
BM(3)-MMC2	463×370 et	0,125	BM	–	–
	1390×1110	4,608	MMC (2 passes)		
BM(3)-MMC4	463×370 et	0,121	BM	–	–
	1390×1110	9,102	MMC (4 passes)		
CensusMMC2	463×370 et 1390×1110	4,338	MMC (2 passes)	107,863	Census
CensusMMC4	463×370 et 1390×1110	8,583	MMC (4 passes)	103,299	Census

5.8 Amélioration de la mise en correspondance par bloc avec la lumière structurée et colorée

Parce que la mise en correspondance par bloc a des difficultés à traiter les régions sans motif, nous avons employé quelques motifs de lumière structurée pour aider la mise en correspondance par bloc. Pour éviter de la lumière forte, nous avons employé une tête de mannequin sur qui nous avons projeté des motifs. La scène a été capturée par l'appareil photo numérique Snap HD mobile MHS-FS3 Bloggie™ de Sony®. La Figure 5.22 en montre un exemple. La disparité maximale de l'image à droite est 15, la carte a été augmentée à l'échelle de 16 pour un meilleur affichage. Nous avons également détiré les images et les cartes de disparité à environs un tiers sur la hauteur pour la raison d'affichage parce que les images originales produites par l'appareil photo MHS-FS3 sont déformées horizontalement.



motif | image de gauche | carte de disparité, la Méthode 2 + remplissage avec les disparités des images 2 fois plus grandes, toutes images sont en couleur

Figure 5.22 Comparaison de projection de lumière naturelle et projection de lumière structurée sur une tête de mannequin

Pour obtenir un bon résultat, le motif de la lumière structurée doit être composé de lignes fines colorées bien visibles, et qui ne changent pas trop entre les lignes adjacentes. Cela devient un problème si la lumière forte est projetée sur la tête humaine. Toutefois, si c'est dans un petit environnement contrôlé, par exemple dans un casque, une lumière faible ou moins forte peut très bien donner le motif sur une tête humaine. Les lignes colorées d'un motif

doivent être fines pour éviter l'ajout de bruit sur la mise en correspondance. Le traitement en nuances de gris peut avoir quelques lignes verticales résultant de la mauvaise correspondance sur la carte de disparité.

Il existe des recherches similaires. Si $G_{\max}$ désigne l'intensité maximale par canal de couleur, Koschan *et al.* ont présenté un spectre de couleurs S_{RVB} de longueur n défini par l'équation 5.6 :

$$\begin{aligned} S_R &= \sin(\frac{i}{n}\pi)(\frac{G_{\max}}{2} - 1) + \frac{G_{\max}}{2}, \\ S_V &= \sin((\frac{2}{3} + \frac{i}{n})\pi)(\frac{G_{\max}}{2} - 1) + \frac{G_{\max}}{2}, i = \{0, \ldots, n\}, \\ S_B &= \sin((\frac{4}{3} + \frac{i}{n})\pi)(\frac{G_{\max}}{2} - 1) + \frac{G_{\max}}{2} \end{aligned} \tag{5.6}$$

Notre codage de lumière est une répétition d'une liste de couleurs aléatoire ou suivant une règle définie. La recherche pour un meilleur codage est hors l'intérêt de ce mémoire.

CHAPITRE 6

CONCLUSION

De nos travaux, nous avons montré le degré d'augmenter la performance de calcul, le potentiel d'améliorer la qualité à l'aide de GPGPU et des options supplémentaires. Nous avons introduit de nouvelles façons d'employer des images en haute définition pour améliorer une plus petite carte de disparité. Bien que la méthode soit exigeante quant aux ressources matérielles, l'emploi de GPGPU nous permet d'en accélérer le travail et d'en réduire le coût.

Nous avons introduit une méthode simple d'interpolation MoyenPlusPropagationDuFond pour efficacement remplir des zones invalides. Nous avons également montré d'autres options incluant l'enlèvement de taches, le filtre bilatéral, la BilSub, le remplissage des disparités des images en plus haute définition, etc. Une bonne combinaison de méthodes peut aider à avoir un bon résultat.

Il existe des méthodes de remplissage des zones où les valeurs de disparité ne sont pas fiables. Toutes ces méthodes procèdent selon quelques hypothèses ou observations. Tout changement de condition altère le fonctionnement de la méthode utilisée. Notre méthode est plus générale. Les résultats de sa mise en épreuve montrent qu'elle fonctionne bien.

Nous avons créé une combinaison de Census et de mise en correspondance par bloc pour explorer le potentiel et montré sa bonne performance, quelques utilités et de contraintes. Il est difficile de déterminer la fonction de combinaison pour la combinaison de Census et de mise en correspondance par bloc. La consistance entre les cartes de disparité de deux directions peut détériorer en fonction de la fonction de combinaison.

Comme solution probabiliste, nous avons employé le MMC et d'autres solutions possibles. Nous avons testé la combinaison de Census et de MMC, la combinaison de mise en correspondance par bloc et de MMC. L'utilisation de plusieurs MMC est similaire à la méthode de propagation de croyance. Le MMC peut aider à s'établir à un état stable. La détermination des matrices de transition et d'émission est essentielle, il faut des environnements similaires pour réussir l'emploi de MMC pour le calcul de disparité. La combinaison de Census et de MMC fonctionne mieux que la combinaison de mise en correspondance par bloc et de MMC.

Notre implémentation de MMC de GPGPU est rapide. Une application sous des contraintes peut avoir une gamme de recherche beaucoup plus petite que prévu. Une bonne analyse peut aider à accélérer encore l'application. Nous pouvons aussi constater que différentes implémentations de l'algorithme Viterbi existent. Il faut donc traiter ce sujet avec soin pour éviter des confusions.

Pour encore améliorer la qualité de la mise en correspondance par bloc, nous avons proposé la méthode basée sur la lumière colorée et structurée afin d'améliorer la performance.

6.1 Synthèse des travaux

Les images en plus haute définition sont plus précises. Toutefois, elles demandent plus de temps dans le traitement, et des détails peuvent créer un artefact. Nous avons étudié quelques méthodes de traitement possibles. Une correspondance simple par bloc peut très bien fonctionner si on ajoute quelques étapes supplémentaires. Le GPGPU peut accélérer le traitement. Nous avons montré que parfois les méthodes simples peuvent travailler aussi bien si on les combine avec quelques suppléments.

Nous avons aussi étudié d'autres méthodes existantes. Par exemple la méthode BilSub ou Census. La méthode BilSub peut être employée pour éliminer trop de variations locales. La méthode Census peut fonctionner avec une certaine robustesse. Une combinaison de plusieurs méthodes s'avère efficace, par exemple, une combinaison de Census et de mise en correspondance par bloc, une combinaison de correspondance et de MMC, une combinaison de Census et de MMC.

6.2 Limitations de la solution proposée

Notre méthode basée sur les disparités des images en plus haute définition ne peut pas être employée là où on demande strictement le temps réel, parce que plusieurs passes demandent toujours plus de temps. Dépendant l'exigence de temps, l'emploi du matériel approprié peut rendre possible le temps réel. Partiellement à cause de notre limite de trouver une bonne fonction de combinaison et le problème de consistance de disparité, la méthode de remplissage des disparités en employant les images en plus haute définition ne fonctionne pas bien pour la méthode de combinaison de Census et de mise en correspondance par bloc.

L'interpolation se base sur une hypothèse, il faut définir un groupe de paramètres pour mieux le faire fonctionner. Donc, si le type d'images change, il faut redéfinir les paramètres.

Le MMC est limité dans l'emploi parce qu'il exige beaucoup de ressources. Le temps d'exécution varie selon les images traitées. Nous supposons qu'il faut aussi déterminer le type d'environnement afin d'avoir un bon groupe des matrices de transition et d'émission. Malheureusement, nous n'avons pas pu approfondir dans cette recherche.

6.3 Améliorations futures

Nous devons continuellement améliorer la méthode de création des cartes de disparité afin d'en augmenter la qualité et la performance. Une implémentation améliorée, des études approfondies de Census sur GPGPU, une recherche sur le potentiel de Census, un emploi des caractéristiques des images, une analyse des structures d'images et des méthodes probabilistes sont des voies possibles pour améliorer la qualité de carte de disparité. Une segmentation d'images et l'emploi de *superpixel*s (voir Ren et Malik, 2003) peuvent aussi contribuer à identifier de petites zones et à éviter des variations locales. Nous devons aussi approfondir les méthodes hiérarchiques afin de tirer le maximum de ces méthodes. Ce que nous avons fait est un peu le contraire de celles traditionnelles, nous devons étudier les avantages et les inconvénients, cherche le potentiel.

Nous pouvons aussi étudier des méthodes de propagation de croyance, ou même envisager d'employer des méthodes d'intelligence artificielle avec le GPGPU. Une généralisation de MMC introduite dans la sous-section 3.13.4 peut augmenter la précision, mais il y a de problèmes des bordures d'objet où des bordures sont plus lisses que souhaité, donc il faut ajouter d'éléments dans le modèle. La puissance de GPGPU peut rendre possibles des méthodes plus complexes dans le calcul des cartes de disparité.

Le modèle de représentation des couleurs CIELab est bon sur le plan de la simulation de la perception humaine. Toutefois, nous cherchons à créer une carte de disparité réelle et pas seulement d'après la perception humaine. Bien que nous pensions que ce n'est pas nécessaire d'employer CIELab, pour mettre à jour les limites de CIELab, il faut étudier la mise en correspondance des pixels dans l'espace CIELab ou même étudier d'autres espaces de couleur.

RÉFÉRENCES

AIRY, G. B. (1834). *Transactions of the cambridge philosophical society*, vol. 5. Cambridge : University Press.

ANSAR, A., CASTANO, A. et MATTHIES, L. (2004). Enhanced real-time stereo using bilateral filtering. *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on*. pp. 455–462.

ARISTOTLE (IVe siècle avant J.-C.). *Problems, books I-XXI, with an english translation by W. S. Hett, M.A.* Harvard University Press / William Heinemann Ltd.

BAKER, J. (1975). The DRAGON system–an overview. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 23, pp. 24–29.

BATLLE, J., MOUADDIB, E. M. et SALVI, J. (1998). Recent progress in coded structured light as a technique to solve the correspondence problem : a survey. *Pattern Recognition*, 31, pp. 963–982.

BIRCHFIELD, S. et TOMASI, C. (1998). Depth discontinuities by pixel-to-pixel stereo. *Computer Vision, 1998. Sixth International Conference on*. pp. 1073–1080.

BLASIAK, A., WEHRWEIN, J. et SCHARSTEIN, D. (2005). 2005 stereo datasets with ground truth. `http://vision.middlebury.edu/stereo/data/scenes2005/`.

BLEYER, M. et GELAUTZ, M. (2005). A layered stereo matching algorithm using image segmentation and global visibility constraints. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59, pp. 128–150.

BOUGUET, J.-Y. (2010). Camera calibration toolbox for MATLAB. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, la dernière visite a eu lieu le 7 juillet 2011.

BRADSKI, G. et KAEHLER, A. (2008). *Learning OpenCV : computer vision with the OpenCV library*. O'Reilly Media.

BROWN, M., BURSCHKA, D. et HAGER, G. (2003). Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25, pp. 993–1008.

COCHRAN, S. et MEDIONI, G. (1992). 3-D surface description from binocular stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14, pp. 981–994.

COMANICIU, D. et MEER, P. (2002). Mean shift : a robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24, 603 –619.

DONATE, A., LIU, X. et COLLINS, E. (2011). Efficient path-based stereo matching with subpixel accuracy. *Systems, Man, and Cybernetics, Part B : Cybernetics, IEEE Transactions on*, 41, pp. 183–195.

DUDA, R. O. et HART, P. E. (1973). *Pattern classification and scene analysis*. John Wiley & Sons, Inc.

DURBIN, R., EDDY, S. R., KROGH, A. et MITCHISON, G. (1999). *Biological sequence analysis - probabilistic models of proteins and nucleic acids*. Cambridge University Press.

ERUHIMOV, V. (2010). Stereo Match. Example of OpenCV.

FAUGERAS, O., HOTZ, B., MATHIEU, H., VIVILLE, T., ZHANG, Z., FUA, P., THRON, E., MOLL, L., BERRY, G., VUILLEMIN, J., BERTIN, P. et PROY, C. (1993). Real time correlation-based stereo : algorithm, implementations and application. *INRIA*.

FELZENSZWALB, P. et HUTTENLOCHER, D. (2004). Efficient belief propagation for early vision. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. vol. 1, pp. 261–268.

FORNEY, G.D., J. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61, pp. 268–278.

FORSYTH, D. A. et PONCE, J. (2002). *Computer vision : a modern approach*. Prentice Hall, Pearson Educations, Inc.

FRISIUS, R. G. (1557). *De radio astronomico et geometrico liber*. ap. Gul Cavellat.

FUA, P. (1991). Combining stereo and monocular information to compute dense depth maps that preserve depth discontinuities. *In Proceedings of the 12th International Joint Conference on Artificial Intelligence*. pp. 1292–1298.

FUKUNAGA, K. et HOSTETLER, L. (1975). The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21, pp. 32–40.

GEIGER, A., ROSER, M. et URTASUN, R. (2010). Efficient large-scale stereo matching. *Asian Conference on Computer Vision*. Queenstown, New Zealand.

GERNSHEIM, H. (1982). *The origins of photography. With 191 illustration*. Thames & Hudson, N.Y.

GIBSON, J. J. (1950). *The perception of the visual world*. Houghton Mifflin.

GLASKOWSKY, P. N. (2009). NVIDIA's Fermi : the first complete GPU computing architecture. `http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf`, la dernière visite a eu lieu le 22 septembre 2011.

GREPSTAD, J. (2011). Pinhole photography – history, images, cameras, formulas. `http://home.online.no/~gjon/pinhole.htm`, la dernière visite a eu lieu le 23 décembre 2011.

HALATCI, I., BROOKS, C. et IAGNEMMA, K. (2007). Terrain classification and classifier fusion for planetary exploration rovers. *Aerospace Conference, 2007 IEEE*. 1 –11.

HAMMOND, J. H. (1981). *The camera obscura. A chronicle.* Adam Hilger Ltd.

HERSCHEL, J. F. W. (1827–1830). *Treatises on physical astronomy, light and sound. Contributed to the Encyclopaedia Metropolitana*. London and Glasgow : Richard Griffin and Company.

HIRSCHMÜLLER, H. (2001). Improvements in real-time correlation-based stereo vision. *Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on*. pp. 141–148.

HIRSCHMÜLLER, H. (2003). *Stereo vision based mapping and immediate virtual walkthroughs*. Thèse de doctorat, De Montfort University.

HIRSCHMÜLLER, H. (2008). Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30, pp. 328–341.

HIRSCHMULLER, H. et GEHRIG, S. (2009). Stereo matching in the presence of sub-pixel calibration errors. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. pp. 437–444.

HIRSCHMÜLLER, H., INNOCENT, P. R. et GARIBALDI, J. (2002). Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47, pp. 229–246.

HIRSCHMÜLLER, H. et SCHARSTEIN, D. (2009). Evaluation of stereo matching costs on images with radiometric differences. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31, pp. 1582–1599.

HORAUD, R. et MONGA, O. (1995). *Vision par ordinateur : outils fondamentaux*. Traité des nouvelles technologies : Série Informatique. Hermès.

HUMENBERGER, M., ENGELKE, T. et KUBINGER, W. (2010). A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality. *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. pp. 77–84.

ISGRO, F., TRUCCO, E., KAUFF, P. et SCHREER, O. (2004). Three-dimensional image processing in the future of immersive media. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, pp. 288–303.

KIM, J., KOLMOGOROV, V. et ZABIH, R. (2003). Visual correspondence using energy minimization and mutual information. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. vol. 2, pp. 1033–1040.

KLEIN, M. V. (1970). *Optics*. John Wiley & Sons, inc.

KOLMOGOROV, V. et ZABIH, R. (2001). Computing visual correspondence with occlusions using graph cuts. *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*. vol. 2, pp. 508–515.

KONOLIGE, K. (1997). Small vision systems : hardware and implementation. *Eighth International Symposium on Robotics Research*. pp. 111–116.

KONOLIGE, K. (2010). Projected texture stereo. *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. pp. 148–155.

KOSCHAN, A., RODEHORST, V. et SPILLER, K. (1996). Color stereo vision using hierarchical block matching and active color illumination. *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. vol. 1, pp. 835–839.

KSCHISCHANG, F., FREY, B. et LOELIGER, H.-A. (2001). Factor graphs and the sum-product algorithm. *Information Theory, IEEE Transactions on*, 47, pp. 498–519.

LAMBRECHT, R. W. et WOODHOUSE, C. (2010). *Way beyond monochrome 2e : advanced techniques for traditional black & white photography including digital negatives and hybrid printing*. Focal Press.

LEE, S. H. et SHARMA, S. (2011). A real-time disparity estimation algorithm for stereo camera systems. *Consumer Electronics, IEEE Transactions on*, 57, pp. 1018–1026.

LIM, J. (2009). Optimized projection pattern supplementing stereo systems. *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*. pp. 2823–2829.

MARR, D. (1982). *Vision : a computational investigation into the human representation and processing of visual information*. W.H. Freeman and Company.

MATTHIES, L., KELLY, A., LITWIN, T. et THARP, G. (1995). Obstacle detection for unmanned ground vehicles : a progress report. Springer-Verlag, pp. 475–486.

MATTHIES, L., MAIMONE, M., JOHNSON, A., CHENG, Y., WILLSON, R., VILLALPANDO, C., GOLDBERG, S., HUERTAS, A., STEIN, A. et ANGELOVA, A. (2007).

Computer vision on Mars. *International Journal of Computer Vision*, 75, pp. 67–92. 10.1007/s11263-007-0046-z.

MATTOCCIA, S. (2009). Stereo vision : algorithms and applications. `http://www.vision.deis.unibo.it/smatt/Seminars/StereoVision.pdf`, la dernière visite a eu lieu le 28 octobre 2011.

MEI, X., SUN, X., ZHOU, M., JIAO, S., WANG, H. et ZHANG, X. (2011). On building an accurate stereo matching system on graphics hardware. *GPUCV 2011*.

MO, D. (Ve siècle avant J.-C.). *Mozi*, vol. 10. En chinois, de traductions existent.

MOHAN, S., SIMONSEN, K., BALSLEV, I., KRUGER, V. et ERIKSEN, R. (2011). 3D scanning of object surfaces using structured light and a single camera image. *Automation Science and Engineering (CASE), 2011 IEEE Conference on*. pp. 151–156.

MORAVEC, H. (1980). Obstacle avoidance and navigation in the real world by a seeing robot rover. *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University*, no. CMU-RI-TR-80-03.

MORAVEC, H. P. (1977). Towards automatic visual obstacle avoidance. *International Joint Conference on Artificial Intelligence*. 584.

MOUADDIB, E., BATLLE, J. et SALVI, J. (1997). Recent progress in structured light in order to solve the correspondence problem in stereovision. *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. vol. 1, pp. 130–136.

MUNSHI, A. (2009). The OpenCL specification, version 1.0.

MURPHY, K. et DUNHAM, M. (2011). Probabilistic modeling toolkit for Matlab/Octave, version 3. `http://code.google.com/p/pmtk3/`, la dernière visite a eu lieu le 30 octobre 2011.

MURRAY, D. et LITTLE, J. (2000). Using real-time stereo vision for mobile robot navigation. *Autonomous Robots*. pp. 161–171.

NEVATIA, R. (1982). *Machine perception*. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632.

NVIDIA (2009). NVIDIA's next generation CUDA compute architecture : Fermi™. `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`, la dernière visite a eu lieu le 22 septembre 2011.

NVIDIA (2011a). CUDA C best practices guide.

NVIDIA (2011b). NVIDIA CUDA C programming guide version 4.0.

PARZEN, E. (1962). On estimation of a probability density function and mode. *Annals of mathematical statistics*. vol. 33, pp. 1065–1076.

PISSALOUX, E., VELAZQUEZ, R. et MAINGREAUD, F. (2008). Intelligent glasses : a multimodal interface for data communication to the visually impaired. *Multisensor fusion and integration for intelligent systems, 2008. MFI 2008. IEEE International Conference on*. pp. 120–124.

PRATT, W. K. (2001). *Digital image processing - PIKS inside, third Edition*. Wiley-Interscience.

RABINER, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, pp. 257–286.

REN, X. et MALIK, J. (2003). Learning a classification model for segmentation. *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. vol. 1, pp. 10–17.

RENNER, E. (2008). *Pinhole photography : from historic technique to digital application*. Focal Press.

ROSENDAHL, S. (2010). Presentation for T-106.5800 seminar on GPGPU programming. `https://wiki.aalto.fi/display/GPGPUK2010/talk+Cuda+and+OpenCL+API+comparison`, la dernière visite a eu lieu le 21 septembre 2011.

SCHARSTEIN, D. et SZELISKI, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47, pp. 7–42.

SCHARSTEIN, D., SZELISKI, R. et ZABIH, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on*.

SCHARSTEIN, D., VANDENBERG-RODES, A. et SZELISKI, R. (2003). 2003 stereo datasets with ground truth. `http://vision.middlebury.edu/stereo/data/scenes2003`, la dernière visite a eu lieu le 30 octobre 2011.

SHANNON, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, vol. 27, pp. 379–423,623–656.

SHIRAI, Y. (1972). Recognition of polyhedrons with a range finder. *Pattern Recognition*, 4, pp. 243–244.

SHIRAI, Y. et SUWA, M. (1971). Recognition of polyhedrons with a range finder. *IJCAI'71, Proceedings of*. pp. 80–87.

SIZINTSEV, M., KUTHIRUMMAL, S., SAMARASEKERA, S., KUMAR, R., SAWHNEY, H. S. et CHAUDHRY, A. (2010). GPU accelerated realtime stereo for augmented reality. *Proceedings of the 5th International Symposium 3D Data Processing, Visualization and Transmission (3DPVT'10)*.

SNYDER, W. E. et QI, H. (2004). *Machine vision*. Cambridge University Press.

STAM, J. (2008). Stereo imaging with CUDA. *http://sourceforge.net/projects/openvidia/files/CUDAStereoCamera/StereoCamerav1.0b/StereoImaging.pdf*.

STRUTT, J. W. (1891). On pin-hole photography. *Philosophical Magazine*, XXXI, pp. 87–99.

STRUTT, J. W. (1902). *Scientific papers by John William Strutt, Baron Rayleigh*, vol. III. C. J. Clay and Sons & Cambridge University Press Warehouse.

SUN, C. (2002). Fast stereo matching using rectangular subregioning and 3D maximum-surface techniques. *Int. J. Comput. Vision*, 47, pp. 99–117.

SZELISKI, R. et ZABIH, R. (1999). An experimental comparison of stereo algorithms. *Vision Algorithms : Theory and Practice, number 1883 in LNCS*. Springer-Verlag, 1–19.

TERZOPOULOS, D. (1982). Multi-Level reconstruction of visual surfaces : variational principles and finite element Representations. Rapport technique, Massachusetts Insititute of Technology.

TERZOPOULOS, D. (1988). The computation of visible-surface representations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 10, pp. 417–438.

THRUN, S. (2011). Solving for depth. `https://www.ai-class.com/course/video/videolecture/201`, la dernière visite a eu lieu le 08 février 2012.

TOMASI, C. et MANDUCHI, R. (1998). Bilateral filtering for gray and color images. *Computer Vision, 1998. Sixth International Conference on*. pp. 839–846.

TRUCCO, E. et VERRI, A. (1998). *Introductory techniques for 3-D computer vision*. Prentice-Hall, Inc.

VIOLA, P. et WELLS, W.M., I. (1995). Alignment by maximization of mutual information. *Computer Vision, 1995. Proceedings., Fifth International Conference on*. pp. 16–23.

VITERBI, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13, pp. 260–269.

WEBER, M., HUMENBERGER, M. et KUBINGER, W. (2009). A very fast census-based stereo matching implementation on a graphics processing unit. *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. pp. 786–793.

WEGNER, P. (1960). A technique for counting ones in a binary computer. *Commun. ACM*, 3, p. 322.

WEGNER, P. (2011). Hamming distance. `http://en.wikipedia.org/wiki/Hamming_distance`, la dernière visite a eu lieu le 17 octobre 2011.

WOODFILL, J. et VON HERZEN, B. (1997). Real-time stereo vision on the PARTS reconfigurable computer. *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*. pp. 201–210.

XU, G. et ZHANG, Z. (1996). *Epipolar geometry in stereo, motion, and object recognition : a unified approach*. Springer, Norwell, MA, USA.

YANG, Q., WANG, L. et AHUJA, N. (2010). A constant-space belief propagation algorithm for stereo matching. *CVPR*.

YANG, Q., WANG, L., YANG, R., STEWENIUS, H. et NISTER, D. (2009). Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31, pp. 492–504.

ZABIH, R. et WOODFILL, J. (1994). Non-parametric local transforms for computing visual correspondence. *Proceedings of the Third European Conference-Volume II on Computer Vision*. Springer-Verlag, London, UK, UK, ECCV '94, pp. 151–158.

ZALEVSKY, Z., ALEXANDER, S., AVIAD, M. et JAVIER, G. (2007). Optical mapping system for e.g. object reconstruction, has imaging device detecting light response of illuminated region and generating image data, which is indicative of object with projected speckles pattern. WIPO Patent WO2007043036 filed 14/03/2006, issued 19/04/2007.

ZHANG, K., LU, J. et LAFRUIT, G. (2009). Cross-based local stereo matching using orthogonal integral images. *Circuits and Systems for Video Technology, IEEE Transactions on*, 19, pp. 1073–1079.

ZINNER, C., HUMENBERGER, M., AMBROSCH, K. et KUBINGER, W. (2008). An optimized software-based implementation of a Census-based stereo matching algorithm. *Proceedings of the 4th International Symposium on Advances in Visual Computing*. Springer-Verlag, Berlin, Heidelberg, ISVC '08, pp. 216–227.

ZITNICK, C. et KANADE, T. (2000). A cooperative algorithm for stereo matching and

occlusion detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22, pp. 675–684.

ANNEXE A

STÉNOPÉ

Un sténopé est un dispositif optique sans lentille permettant d'obtenir des images photographiques sous la forme d'une chambre noire. Il s'agit d'une ouverture de très faible diamètre. Le phénomène a été observé en Chine durant le Ve siècle avant J.-C. par Mo Di (aussi appelé Mozi, ou Mo Tzu) (voir Mo, Ve siècle avant J.-C., Jingshuoxia N^{o} 43). Hammond a aussi noté ce fait (voir Hammond, 1981).

Aristote au IVe siècle avant J.-C. se demandait pourquoi la lumière du soleil passant à travers les quadrilatères, par exemple, l'un des trous dans la vannerie ne crée pas une image rectangulaire, mais plutôt un rond, et pourquoi l'image de l'éclipse solaire passant à travers un tamis, les feuilles d'un arbre ou les écarts entre les doigts croisés crée un croissant sur la terre. (voir Aristotle, IVe siècle avant J.-C., Book XV)

En 1425, Filippo Brunelleschi a employé le sténopé pour observer une éclipse solaire. (voir Renner, 2008) La première illustration publiée de sténopé en 1544 est montrée dans la Figure A.1 (voir Gernsheim, 1982, p. 9).

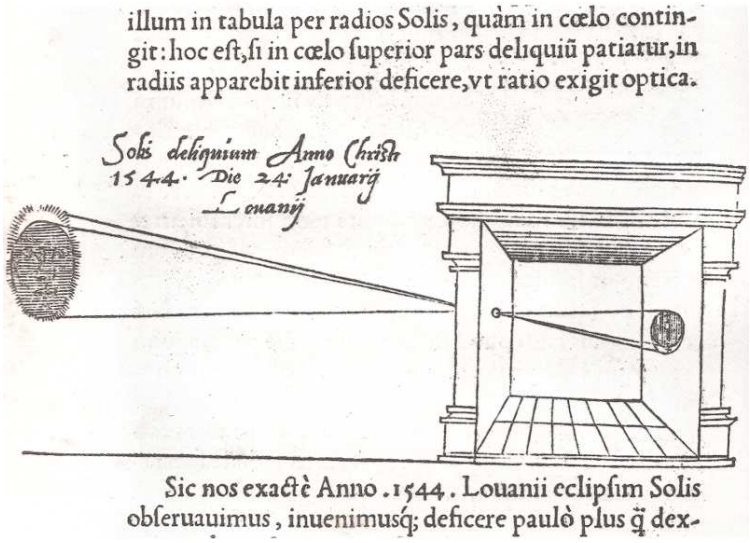


Figure A.1 La première illustration publiée de sténopé : observation d'éclipse solaire en janvier 1544 à Louvain par Reinier Gemma Frisius, Collection Gernsheim (voir Gernsheim, 1982, p. 9)

Une figure similaire a été publiée dans le livre de Frisius en 1557 (voir Frisius, 1557, p. 39).

Figure A.2 L'illustration publiée de sténopé : observation d'éclipse solaire en janvier 1544 à Louvain par Reinier Gemma Frisius (voir Frisius, 1557, p. 39)

Grepstad a aussi présenté quelques matériaux concernant le sténopé (voir Grepstad, 2011).

Figure d'Airy

Un trou trop grand ou trop petit n'améliore pas l'image capturée. Un trou trop grand va laisse passer plus de lumière qui rend la scène floue, tandis qu'un trou trop petit laisse entrer trop peu de lumière pour bien former la scène. Pour connaître le diamètre optimal de sténopé, nous devons d'abord connaître la figure d'Airy.

La lumière compose des ondes électromagnétiques visibles. Elle se déplace en ligne droite dans un milieu transparent homogène. Toutefois, la nature ondulatoire de la lumière fait que celle-ci est diffractée si elle est matériellement limitée, par exemple au passage à travers un trou. Si le trou est parfaitement circulaire, la figure de diffraction est appelée figure d'Airy (du nom de George Biddell Airy), présentant un disque central primaire et des cercles concentriques secondaires de plus en plus atténués. En 1834, Airy a présenté une analyse mathématique de ce phénomène, d'où vient l'appellation de figure d'Airy. (voir Airy, 1834, p. 283–290)

Selon John Herschel, Sir William Herschel a été le premier qui a observé le phénomène. John Herschel a décrit le phénomène en détail vers 1827. Il a décrit le point de lumière comme : (voir HERSCHEL, 1827–1830, p. 491) "...un disque planétaire parfaitement rond et bien défini, entouré par deux, trois ou plusieurs anneaux alternativement sombres et lumineux, qui, si on les examine attentivement, sont perçus comme légèrement colorés à leurs frontières. Ils se succèdent de près à des intervalles égaux autour du disque central, et sont

généralement beaucoup mieux vus et plus régulièrement et parfaitement formés dans les télescopes réfractants que ceux reflétants (*a perfectly round, well-defined planetary disc, surrounded by two, three, or more alternately dark and bright rings, which, if examined attentively, are seen to be slightly coloured at their borders. They succeed each other nearly at equal intervals round the central disc, and are usually much better seen and more regularly and perfectly formed in refracting than in reflecting telescopes*)..." (voir HERSCHEL, 1827–1830, p. 491)

Lorsqu'une onde traverse une ouverture ou autour d'un objet, l'interférence entre l'onde et l'objet va produire le phénomène de diffraction. Le principe de Huygens dit que chaque partie de la perturbation optique sur une surface de référence dans l'espace agit comme une source d'ondelettes sphériques qui se combinent pour donner la perturbation à un point éloigné. On va employer l'approche de Fresnel pour étudier la superposition. (voir Klein, 1970, p. 293)

La surface intégrale d'un trou dans un plan peut être calculée par (voir Klein, 1970, p. 296)

$$\tilde{E}(r) = \frac{i}{\lambda} \int\int_{\sum_0} \tilde{E}_{inc}(r') \frac{e^{-ik|r-r'|}}{|r-r'|} d\sigma \tag{A.1}$$

où $d\sigma$ représente un élément infinitésimal de la surface au point r' à l'ouverture $\sum_0$, $\tilde{E}_{inc}(r')$ est le champ incident au point r', λ est la longueur d'onde de la lumière. (voir Klein, 1970, p. 296)

Un phénomène spécial s'appelant la diffraction de Fraunhofer aura lieu quand le chemin optique allant de points dans l'ouverture de diffraction au point d'observation est déterminé linéairement par les coordonnées du point d'ouverture (voir Klein, 1970, p. 306). Basé sur l'équation A.1, nous pouvons employer une fonction de transmission $t(r') = t(x', y')$ pour obtenir une généralisation dans l'équation A.2 (voir Klein, 1970, p. 308).

$$\tilde{E}(r) = \frac{i}{\lambda} \tilde{E}(O) \frac{e^{-ikR_0'}}{R_0'} \int\int_{-\infty}^{\infty} t(x', y') e^{-2\pi i(ux' - vy')} dx' dy' \text{ où } u = (\alpha - \alpha')/\lambda, v = (\beta - \beta')/\lambda \tag{A.2}$$

α et β sont deux coordonnées des cosinus directeurs de la lumière incidente, α' et β' sont celles des cosinus directeurs de la lumière de diffraction.

Pour une ouverture circulaire comme dans la Figure A.3, r' et σ sont les coordonnées polaires dans l'ouverture. Nous pouvons avoir l'équation A.3. (voir Klein, 1970, p. 317–318).

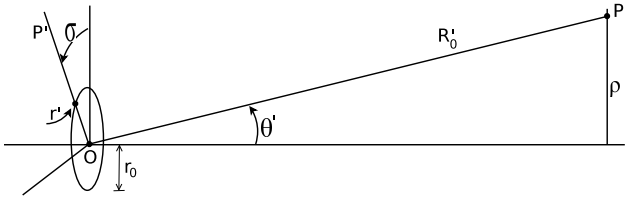


Figure A.3 Coordinates dans diffraction Fraunhofer (voir Klein, 1970, p. 317)

$$\tilde{E}(r) = \frac{i}{\lambda}\tilde{E}(O)\frac{e^{-ikR_0^{'}}}{R_0^{'}}ikr_0^2\frac{J_1(kr_0\sin\theta')}{kr_0\sin\theta'} \tag{A.3}$$

En posant $t = kr_0\sin\theta' = \dfrac{2\pi r_0\rho}{(\lambda R_0^{'})} = 2\pi r_0 w$, où $w = \sqrt{u^2+v^2} = \sqrt{\dfrac{\alpha'^2+\beta'^2}{\lambda^2}}$, on obtient l'équation A.4 (voir Klein, 1970, p. 318) :

$$\tilde{E}(r) = \tilde{E}(O)\frac{e^{-ikR_0^{'}}}{R_0^{'}}ikr_0^2\frac{J_1(t)}{t} \tag{A.4}$$

La fonction $J_1(t)$ est appelée la fonction de Bessel de premier ordre. Le premier disque maximal est donné quand $t = 1,22\pi = 3,83$. (voir Klein, 1970, p. 318–319) t est exprimé en radius, nous avons la valeur $1,22$ comme le facteur de radius. En conséquence, la valeur $2,44$ est employée comme le facteur de diamètre.

Diamètre optimal de sténopé

Le diamètre optimal de sténopé à une certaine distance limitée par la figure d'Airy est calculé par la diffraction de Fraunhofer. Lord Rayleigh a indiqué que le diamètre de l'ouverture optimal peut être calculé avec la relation $d^2 = f\lambda$, où d est le diamètre de l'ouverture, f est la longueur focale et λ est la longueur d'onde en 1891. (voir Strutt, 1891) (voir Strutt, 1902, p. 436) Maintenant, en considérant l'effet de la figure d'Airy, nous pouvons calculer le diamètre minimal par l'équation A.5. (voir Lambrecht et Woodhouse, 2010, p. 154)

$$d = \sqrt{2,44\lambda f} \tag{A.5}$$

Par le critère d'interférence de Rayleigh qui prévoit que l'on ne peut séparer au maximum deux images que quand le maximum de la figure d'Airy d'une image se superpose au premier minimum de l'autre. Nous pouvons aussi calculer le diamètre maximal par l'équation A.6. (voir

Lambrecht et Woodhouse, 2010, p. 154)

$$d = \sqrt{3,66\lambda f} \tag{A.6}$$

Une valeur couramment employée pour la longueur d'onde λ est 555 nm pour les yeux et la photographie picturale standard (voir Lambrecht et Woodhouse, 2010, p. 155).

Parce que $\sqrt{3,66} \approx 1,91$, on voit souvent que la formule pour déterminer le meilleur diamètre est $d = 1,9\sqrt{f\lambda}$.

ANNEXE B

DIFFÉRENCES ENTRE DEUX IMPLÉMENTATIONS DE L'ALGORITHME VITERBI

Il y a plusieurs définitions et implémentations disponibles de l'algorithme Viterbi. Au cours de nos expériences, nous avons vérifié la différence entre hmmviterbi fournie par MATLAB® et viterbi fournie par PMTK3 (voir Murphy et Dunham, 2011) :

Comparaisons des implémentations de MATLAB et de PMTK3

La fonction hmmviterbi de MATLAB a implémenté l'algorithme décrit par Durbin *et al.*(voir Durbin *et al.*, 1999, p. 56), qui suppose que le modèle est initialement dans l'état 1 à l'étape 0. Tandis que la fonction hmmMap suppose que le modèle est initialement dans les états définis par la distribution initiale et les premières observations. La fonction hmmMap de PMTK3 a ajouté la valeur de précision relative de virgule flottante (*floating-point relative accuracy*) "eps" qui est 2^{-52} pour améliorer la précision.

Concernant la performance, la hmmviterbi de MATLAB doit être un peu plus lente parce qu'elle est en script. Tandis que le PMTK3 supporte les sous-routines liées dynamiquement (*dynamic-link library*), le code sera exécuté en mode MEX compilé de C, qui est plus rapide que hmmviterbi selon notre évaluation. Le taux d'accélération est d'environs 8,93.

Le reste du traitement sera identique pour les deux implémentations.

Notre implémentation ressemble à celle de PMTK3. À cette différence près que nous n'avons pas ajouté la valeur de précision relative de virgule flottante "eps". Il est nécessaire d'ajouter la distribution initiale pour conformer à la nature probabiliste.

Notes sur l'algorithme Viterbi

Selon Viterbi, la probabilité de distribution doit être identique pour chaque élément et pour chaque itinéraire. Les éléments sur un itinéraire précis sont indépendants statistiquement (voir Viterbi, 1967, p. 265). La présentation de l'algorithme Viterbi de Durbin *et al.* suppose qu'initialement le système est dans l'état identifié comme 1 (voir Durbin *et al.*, 1999, p. 56). Tandis que l'implémentation de PMTK3 (voir Murphy et Dunham, 2011) emploie une distribution initiale d'états et des observations initiales. Nous supposons qu'une distribution initiale d'états est plus naturelle. Toutefois, la présentation de Durbin *et al.* ne demandant pas une distribution initiale est plus facile à employer.

Un article fait par Forney a bien expliqué l'algorithme Viterbi. Dans cet article, nous pouvons voir que l'auteur emploie une valeur arbitraire pour l'état initial (voir Forney, 1973, p. 272).

ANNEXE C

CORRECTIONS D'UN EXEMPLE DE SDK DE NVIDIA

Dans l'implémentation de l'algorithme Viterbi livrée dans le kit de développement par NVIDIA® en novembre 2010, nous avons trouvé plusieurs erreurs que nous avons signalées à NVIDIA® le 5 mars 2011 à l'aide d'un formulaire de contact et une adresse courriel de réseau de développeur sans jamais avoir de réponse de sa part.

Erreurs et corrections d'un code de NVIDIA

Dans le programme principal oclHiddenMarkovModel.cpp, la fonction initHMM fournit des probabilités. Toutefois, dans HMM.cpp, on traite ces probabilités en format logarithme, ce qui est une première erreur. Nous avons converti ces données en logarithme.

Ensuite, on doit normalement fournir des probabilités initiales. Or, l'implémentation ne fournit pas cette fonctionnalité. Bien sûr, ce n'est pas une erreur, mais une lacune. Nous avons ajouté cette fonctionnalité.

Dans la classe HMM, l'implémentation n'a pas d'option pour définir le nombre de catégories d'émission. Elle suppose le même nombre des états et des observations. Nous avons ajouté une option pour définir le nombre de catégories d'émission.

Finalement dans le code noyau, on attribue une valeur -1.0f comme la valeur minimale qui est au domaine de probabilité. Cependant, l'opération est une addition qui est au domaine de logarithme. Un problème plus grave se trouve dans l'application de l'algorithme Viterbi. La partie récursive dans le kit de développement est totalement fausse. Nous avons corrigé cette partie. La raison pour laquelle les versions GPGPU et CPU produisent le même résultat est que ces deux versions ont employé de façon erronée le même algorithme, alors que celui-ci est un peu différent de celui de Viterbi.

Observations

Le code fourni par NVIDIA® l'est dans un but de référence, car ce code n'a pas été rigoureusement vérifié. Le même problème existe pour le kit de développement version 4.0 du mai 2011.

Codes corrigés de l'implémentation de l'algorithme Viterbi d'un exemple de SDK de NVIDIA

Ci-dessous nous affichons quelques modifications principales. Les ajouts et les modifications sont soulignés.

Dans la partie initialisation de la classe HMM du fichier oclHiddenMarkovModel.cpp, nous avons ajouté quelques lignes d'initialisation de probabilité et modifié quelques endroits :

```
...
float **initCase = (float**)malloc(nDevice*sizeof(float*));
...
HMM **oHmm = (HMM**)malloc(nDevice*sizeof(HMM*));
for (cl_uint iDevice = 0; iDevice < nDevice; iDevice++) {
   initCase[iDevice] = (float*)malloc(sizeof(float)*nState);
   for (int i = 0; i < nState; i++)
      initCase[iDevice][i] = initProb[i] + mtEmit[i * nEmit + obs[iDevice][0]];
   oHmm[iDevice] = new HMM(cxGPUContext, cqCommandQue[iDevice], initCase[iDevice], mtState, mtEmit,
                           nState, nEmit, nObs, argv[0], wgSize);
}
...
   szWorkGroup = oHmm[iDevice]->ViterbiSearch(vProb[iDevice], vPath[iDevice], obs[iDevice], nEmit);
...
   err = ViterbiCPU(viterbiProbCPU[iDevice], viterbiPathCPU[iDevice], obs[iDevice], nObs, initProb,
                           mtState, nState, mtEmit, nEmit);
...
   free(initCase[iDevice]);
   free(obs[iDevice]);
   free(viterbiPathCPU[iDevice]);
...
```

Dans la fonction ci-dessous du même fichier :

```
int initHMM(float *initProb, float *mtState, float *mtObs, const int &nState, const int &nEmit)
...
for (int i = 0; i < nState; i++) {
   initProb[i] /= sum;
```

```
    initProb[i] = log(initProb[i]);
}
...
    for (int j = 0; j < nState; j++) {
        mtState[i*nState + j] /= RAND_MAX;
        mtState[i*nState + j] = log(mtState[i*nState + j]);
    }
...
    for (int i = 0; i < nEmit; i++) {
        mtObs[i*nState + j] /= sum;
        mtObs[i*nState + j] = log(mtObs[i*nState + j]);
    }
...
```

C'est-à-dire que nous avons transformé toutes les variables initProb, mtState, mtObs en logarithme.

Dans le fichier Viterbi.cl :

```
...
__kernel void ViterbiOneStep(__global float *maxProbNew, __global int *path, __global float *maxProbOld,
                             __global float *mtState, __global float *mtEmit, __local float maxValue[],
                             __local int maxInd[], int nState, int obs, int iObs, int nEmit) {
...
float mValue = -INFINITY;
int mInd = -1;
float value;
for (int i = localId; i < nState; i += localSize) {
    value = maxProbOld[i] + mtState[i*nState + iState];
    if (value > mValue) {
        mValue = value;
        mInd = i;
    }
}
maxValue[localId] = mValue;
```

```
maxInd[localId] = mInd;
barrier(CLK_LOCAL_MEM_FENCE);
maxOneBlock(maxValue, maxInd);
// copy results from local to global memory
if (localId == 0) {
   maxProbNew[iState] = maxValue[0] + mtEmit[iState * nEmit + obs];
   path[(iObs-1)*nState + iState] = maxInd[0]; }
}
__kernel void ViterbiPath(__global float *vProb, __global int *vPath, __global float *maxProbNew,
                          __global int *path, int nState, int nObs) {
// find the final most probable state
if (get_global_id(0) == 0) {
   float maxProb = -INFINITY;
   int maxState = -1;
...
```

Dans le fichier ViterbiCPU.cpp :

```
#include <cstdlib>
#include <cstdio>
#include <limits>
#define MINVALUE -std : :numeric_limits<float> : :infinity()
int ViterbiCPU(float &viterbiProb, int *viterbiPath, int *obs, const int &nObs, float *initProb,
               float *mtState, const int &nState, float *mtEmit, const int &nEmit){
...
// initial probability
for (int i = 0; i < nState; i++) {
   maxProbOld[i] = initProb[i] + mtEmit[i * nEmit + obs[0]];
}
// main iteration of Viterbi algorithm
for (int t = 1; t < nObs; t++) // for every input observation {
   for (int iState = 0; iState < nState; iState++) {
      // find the most probable previous state leading to iState
```

```
        float maxProb = MINVALUE;
        int maxState = -1;
        for (int preState = 0; preState < nState; preState++) {
            float p = maxProbOld[preState] + mtState[preState*nState + iState];
            if (p > maxProb) {
                maxProb = p;
                maxState = preState;
            }
        }
        maxProbNew[iState] = maxProb + mtEmit[iState * nEmit + obs[t]];
        path[t-1][iState] = maxState;
    }
    for (int iState = 0; iState < nState; iState++) {
        maxProbOld[iState] = maxProbNew[iState];
    }
}
// find the final most probable state
float maxProb = MINVALUE;
...
```

Dans le fichier HMM.cpp :

```
...
size_t HMM : :ViterbiOneStep(const int &obs, const int &iObs, const int &nEmit)
...
err |= clSetKernelArg(ckViterbiOneStep, 10, sizeof(int), (void*)&nEmit);
...
size_t HMM : :ViterbiSearch(cl_mem vProb, cl_mem vPath, int *obs, const int &nEmit)
...
for (int iObs = 1; iObs < nObs; iObs++) {
    szWorkgroup = ViterbiOneStep(obs[iObs], iObs, nEmit);
...
```

Également dans le fichier HMM.h :

```
...

size_t ViterbiSearch(cl_mem vProb, cl_mem vPath, int *obs, const int &nEmit);

...

size_t ViterbiOneStep(const int &obs, const int &iObs, const int &nEmit);

...
```